

NETMOD WG  
Internet-Draft  
Intended status: Standards Track  
Expires: December 17, 2017

D. Bogdanovic  
Volta Networks  
M. Jethanandani  
Cisco Systems, Inc  
L. Huang  
General Electric  
S. Agarwal  
Cisco Systems, Inc.  
D. Blair  
Cisco Systems, INC  
June 15, 2017

Network Access Control List (ACL) YANG Data Model  
draft-ietf-netmod-acl-model-11

Abstract

This document describes a data model of Access Control List (ACL) basic building blocks.

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. Please note that no other RFC Editor instructions are specified anywhere else in this document.

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements

- o "XXXX" --> the assigned RFC value for this draft.
- o Revision date in model (Oct 12, 2016) needs to get updated with the date the draft gets approved. The date also needs to get reflected on the line with <CODE BEGINS>.

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 <http://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 December 17, 2017.

#### Copyright Notice

Copyright (c) 2017 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 (<http://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 . . . . .	3
1.1. Definitions and Acronyms . . . . .	3
2. Problem Statement . . . . .	4
3. Understanding ACL's Filters and Actions . . . . .	4
3.1. ACL Modules . . . . .	5
4. ACL YANG Models . . . . .	9
4.1. IETF Access Control List module . . . . .	9
4.2. IETF Packet Fields module . . . . .	18
4.3. An ACL Example . . . . .	28
4.4. Port Range Usage Example . . . . .	28
5. Security Considerations . . . . .	29
6. IANA Considerations . . . . .	30
7. Acknowledgements . . . . .	30
8. Open Issues . . . . .	31
9. References . . . . .	31
9.1. Normative References . . . . .	31
9.2. Informative References . . . . .	32
Appendix A. Extending ACL model examples . . . . .	32
A.1. Example of extending existing model for route filtering . . . . .	32
A.2. A company proprietary module example . . . . .	34
A.3. Example to augment model with mixed ACL type . . . . .	42
A.4. Linux nftables . . . . .	43
Authors' Addresses . . . . .	44

## 1. Introduction

Access Control List (ACL) is one of the basic elements to configure device forwarding behavior. It is used in many networking concepts such as Policy Based Routing, Firewalls etc.

An ACL is an ordered set of rules that is used to filter traffic on a networking device. Each rule is represented by an Access Control Entry (ACE).

Each ACE has a group of match criteria and a group of action criteria.

The match criteria consist of a tuple of packet header match criteria and can have metadata match criteria as well.

- o Packet header matches apply to fields visible in the packet such as address or class of service or port numbers.
- o In case vendor supports it, metadata matches apply to fields associated with the packet but not in the packet header such as input interface or overall packet length

The actions specify what to do with the packet when the matching criteria is met. These actions are any operations that would apply to the packet, such as counting, policing, or simply forwarding. The list of potential actions is endless depending on the innovations of the networked devices.

Access Control List is also widely known as ACL (pronounce as [ak-uh l]) or Access List. In this document, Access Control List, ACL and Access List are used interchangeably.

The matching of filters and actions in an ACE/ACL are triggered only after application/attachment of the ACL to an interface, VRF, vty/tty session, QoS policy, routing protocols amongst various other config attachment points. Once attached, it is used for filtering traffic using the match criteria in the ACE's and taking appropriate action(s) that have been configured against that ACE. In order to apply an ACL to any attachment point, vendors would have to augment the ACL YANG model.

### 1.1. Definitions and Acronyms

ACE: Access Control Entry

ACL: Access Control List

DSCP: Differentiated Services Code Point

ICMP: Internet Control Message Protocol

IP: Internet Protocol

IPv4: Internet Protocol version 4

IPv6: Internet Protocol version 6

MAC: Media Access Control

TCP: Transmission Control Protocol

## 2. Problem Statement

This document defines a YANG [RFC6020] data model for the configuration of ACLs. It is very important that model can be easily used by applications/attachments.

ACL implementations in every device may vary greatly in terms of the filter constructs and actions that they support. Therefore this draft proposes a model that can be augmented by standard extensions and vendor proprietary models.

## 3. Understanding ACL's Filters and Actions

Although different vendors have different ACL data models, there is a common understanding of what access control list (ACL) is. A network system usually have a list of ACLs, and each ACL contains an ordered list of rules, also known as access list entries - ACEs. Each ACE has a group of match criteria and a group of action criteria. The match criteria consist of packet header matching. It is also possible for ACE to match on metadata, if supported by the vendor. Packet header matching applies to fields visible in the packet such as address or class of service or port numbers. Metadata matching applies to fields associated with the packet, but not in the packet header such as input interface, packet length, or source or destination prefix length. The actions can be any sort of operation from logging to rate limiting or dropping to simply forwarding. Actions on the first matching ACE are applied with no processing of subsequent ACEs.

The model also includes a container to hold overall operational state for each ACL and operational state for each ACE. One ACL can be applied to multiple targets within the device, such as interfaces of a networked device, applications or features running in the device, etc. When applied to interfaces of a networked device, the ACL is

applied in a direction which indicates if it should be applied to packet entering (input) or leaving the device (output). An example in the appendix shows how to express it in YANG model.

This draft tries to address the commonalities between all vendors and create a common model, which can be augmented with proprietary models. The base model is simple and with this design we hope to achieve enough flexibility for each vendor to extend the base model. The use of feature statements in the document allows vendors to advertise match rules they support.

### 3.1. ACL Modules

There are two YANG modules in the model. The first module, "ietf-access-control-list", defines generic ACL aspects which are common to all ACLs regardless of their type or vendor. In effect, the module can be viewed as providing a generic ACL "superclass". It imports the second module, "ietf-packet-fields". The match container in "ietf-access-control-list" uses groupings in "ietf-packet-fields". The combination of if-feature checks and must statements allow for the selection of relevant match fields that a user can define rules for.

If there is a need to define new "matches" choice, such as IPFIX [RFC5101], the container "matches" can be augmented.

For a reference to the annotations used in the diagram below, see YANG Tree Diagrams [I-D.ietf-netmod-yang-tree-diagrams].

```

module: ietf-access-control-list
  +--rw access-lists
    +--rw acl* [acl-type acl-name]
      +--rw acl-name          string
      +--rw acl-type          acl-type
      +--ro acl-oper-data
      +--rw access-list-entries
        +--rw ace* [rule-name]
          +--rw rule-name      string
          +--rw matches
            | +--rw l2-acl {l2-acl}?
            | | +--rw destination-mac-address?      yang:mac-ad
dress      | |
dress      | | +--rw destination-mac-address-mask?  yang:mac-ad
dress      | |
dress      | | +--rw source-mac-address?            yang:mac-ad
dress      | |
dress      | | +--rw source-mac-address-mask?      yang:mac-ad

```

```

| | +--rw ether-type? string
+--rw ipv4-acl {ipv4-acl}?
| | +--rw tos? uint8
| | +--rw length? uint16
| | +--rw ttl? uint8
| | +--rw protocol? uint8
| | +--rw source-port-range!
| | | +--rw lower-port inet:port-number
| | | +--rw upper-port? inet:port-number
+--rw destination-port-range!
| | +--rw lower-port inet:port-number
| | +--rw upper-port? inet:port-number
+--rw ihl? uint8
+--rw flags? bits
+--rw offset? uint16
+--rw identification? uint16
+--rw destination-ipv4-network? inet:ipv4-prefi
x
| | +--rw source-ipv4-network? inet:ipv4-prefi
x
+--rw ipv6-acl {ipv6-acl}?
| | +--rw tos? uint8
| | +--rw length? uint16
| | +--rw ttl? uint8
| | +--rw protocol? uint8
| | +--rw source-port-range!
| | | +--rw lower-port inet:port-number
| | | +--rw upper-port? inet:port-number
+--rw destination-port-range!
| | +--rw lower-port inet:port-number
| | +--rw upper-port? inet:port-number
+--rw next-header? uint8
+--rw destination-ipv6-network? inet:ipv6-prefi
x
| | +--rw source-ipv6-network? inet:ipv6-prefi
x
| | +--rw flow-label? inet:ipv6-flow-
label
+--rw l2-l3-ipv4-acl {mixed-ipv4-acl}?
| | +--rw destination-mac-address? yang:mac-ad
dress
| | +--rw destination-mac-address-mask? yang:mac-ad
dress
| | +--rw source-mac-address? yang:mac-ad
dress
| | +--rw source-mac-address-mask? yang:mac-ad
dress
| | +--rw ether-type? string

```

		<pre> +--rw tos?                               uint8 +--rw length?                             uint16 +--rw ttl?                                uint8 +--rw protocol?                           uint8 +--rw source-port-range!     +--rw lower-port      inet:port-number     +--rw upper-port?     inet:port-number +--rw destination-port-range!     +--rw lower-port      inet:port-number     +--rw upper-port?     inet:port-number +--rw ihl?                               uint8 +--rw flags?                             bits +--rw offset?                             uint16 +--rw identification?                     uint16 +--rw destination-ipv4-network?           inet:ipv4-p </pre>
refix		
		<pre> +--rw source-ipv4-network?               inet:ipv4-p </pre>
refix		
		<pre> +--rw l2-l3-ipv6-acl {mixed-ipv6-acl}?     +--rw destination-mac-address?       yang:mac-ad </pre>
dress		
		<pre> +--rw destination-mac-address-mask?     yang:mac-ad </pre>
dress		
		<pre> +--rw source-mac-address?                yang:mac-ad </pre>
dress		
		<pre> +--rw source-mac-address-mask?           yang:mac-ad </pre>
dress		
		<pre> +--rw ether-type?                        string +--rw tos?                               uint8 +--rw length?                             uint16 +--rw ttl?                                uint8 +--rw protocol?                           uint8 +--rw source-port-range!     +--rw lower-port      inet:port-number     +--rw upper-port?     inet:port-number +--rw destination-port-range!     +--rw lower-port      inet:port-number     +--rw upper-port?     inet:port-number +--rw next-header?                       uint8 +--rw destination-ipv6-network?           inet:ipv6-p </pre>
refix		
		<pre> +--rw source-ipv6-network?               inet:ipv6-p </pre>
refix		
		<pre> +--rw flow-label?                         inet:ipv6-f </pre>
low-label		
		<pre> +--rw l2-l3-ipv4-ipv6-acl {l2-l3-ipv4-ipv6-acl}?     +--rw destination-mac-address?       yang:mac-ad </pre>
dress		

dress			+++rw destination-mac-address-mask?	yang:mac-ad
dress			+++rw source-mac-address?	yang:mac-ad
dress			+++rw source-mac-address-mask?	yang:mac-ad
dress			+++rw ether-type?	string
			+++rw tos?	uint8
			+++rw length?	uint16
			+++rw ttl?	uint8
			+++rw protocol?	uint8
			+++rw source-port-range!	
			+++rw lower-port	inet:port-number
			+++rw upper-port?	inet:port-number
			+++rw destination-port-range!	
			+++rw lower-port	inet:port-number
			+++rw upper-port?	inet:port-number
			+++rw ihl?	uint8
			+++rw flags?	bits
			+++rw offset?	uint16
			+++rw identification?	uint16
refix			+++rw destination-ipv4-network?	inet:ipv4-p
refix			+++rw source-ipv4-network?	inet:ipv4-p
refix			+++rw next-header?	uint8
refix			+++rw destination-ipv6-network?	inet:ipv6-p
refix			+++rw source-ipv6-network?	inet:ipv6-p
low-label			+++rw flow-label?	inet:ipv6-f
			+++rw tcp-acl {tcp-acl}?	
			+++rw sequence-number?	uint32
			+++rw acknowledgement-number?	uint32
			+++rw data-offset?	uint8
			+++rw reserved?	uint8
			+++rw flags?	uint16
			+++rw window-size?	uint16
			+++rw urgent-pointer?	uint16
			+++rw options?	uint32
			+++rw udp-acl {udp-acl}?	
			+++rw length?	uint16
			+++rw icmp-acl {icmp-acl}?	
			+++rw type?	uint8
			+++rw code?	uint8
			+++rw rest-of-header?	uint32
			+++rw any-acl! {any-acl}?	



```

+--rw actions
|   +--rw (packet-handling)?
|   |   +--:(deny)
|   |   |   +--rw deny?      empty
|   |   +--:(permit)
|   |   |   +--rw permit?    empty
|   +--rw logging?    boolean
+--ro ace-oper-data
    +--ro match-counter?  yang:counter64

```

## 4. ACL YANG Models

### 4.1. IETF Access Control List module

"ietf-access-control-list" is the standard top level module for access lists. The "access-lists" container stores a list of "acl". Each "acl" has information identifying the access list by a name("acl-name") and a list("access-list-entries") of rules associated with the "acl-name". Each of the entries in the list("access-list-entries"), indexed by the string "rule-name", has containers defining "matches" and "actions".

The "matches" define criteria used to identify patterns in "ietf-packet-fields". The "actions" define behavior to undertake once a "match" has been identified. In addition to permit and deny for actions, a logging option allows for a match to be logged that can be used to determine which rule was matched upon.

<CODE BEGINS> file "ietf-access-control-list@2017-06-16.yang"

```

module ietf-access-control-list {
  namespace "urn:ietf:params:xml:ns:yang:ietf-access-control-list";
  prefix acl;
  import ietf-yang-types {
    prefix yang;
  }
  import ietf-packet-fields {
    prefix packet-fields;
  }
  organization
    "IETF NETMOD (NETCONF Data Modeling Language)
     Working Group";

  contact
    "WG Web: http://tools.ietf.org/wg/netmod/
     WG List: netmod@ietf.org
     Editor: Dean Bogdanovic
     ivandean@gmail.com

```

Editor: Mahesh Jethanandani  
mjethanandani@gmail.com  
Editor: Lisa Huang  
lyihuangl6@gmail.com  
Editor: Sonal Agarwal  
agarwaso@cisco.com  
Editor: Dana Blair  
dblair@cisco.com";

description

"This YANG module defines a component that describing the configuration of Access Control Lists (ACLs).  
Copyright (c) 2016 IETF Trust and the persons identified as the document authors. 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>).  
This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

revision 2017-06-16 {

description

"Added feature and identity statements for different types of rule matches. Split the matching rules based on the feature statement and added a must statement within each container.";

reference

"RFC XXX: Network Access Control List (ACL) YANG Data Model.";

}

revision 2016-10-12 {

description

"Base model for Network Access Control List (ACL).";

reference

"RFC XXXX: Network Access Control List (ACL) YANG Data Model";

}

/\*

\* Identities

\*/

identity acl-base {

description

"Base Access Control List type for all Access Control List type identifiers.";

```
}

identity ipv4-acl {
  base acl:acl-base;
  description
    "ACL that primarily matches on fields from the IPv4 header
    (e.g. IPv4 destination address) and layer 4 headers (e.g. TCP
    destination port). An acl of type ipv4-acl does not contain
    matches on fields in the ethernet header or the IPv6 header.";
}

identity ipv6-acl {
  base acl:acl-base;
  description
    "ACL that primarily matches on fields from the IPv6 header
    (e.g. IPv6 destination address) and layer 4 headers (e.g. TCP
    destination port). An acl of type ipv6-acl does not contain
    matches on fields in the ethernet header or the IPv4 header.";
}

identity eth-acl {
  base acl:acl-base;
  description
    "ACL that primarily matches on fields in the ethernet header,
    like 10/100/1000baseT or WiFi Access Control List. An acl of
    type eth-acl does not contain matches on fields in the IPv4
    header, IPv6 header or layer 4 headers.";
}

identity mixed-l2-l3-ipv4-acl {
  base "acl:acl-base";

  description
    "ACL that contains a mix of entries that
    primarily match on fields in ethernet headers,
    entries that primarily match on IPv4 headers.
    Matching on layer 4 header fields may also exist in the
    list.";
}

identity mixed-l2-l3-ipv6-acl {
  base "acl:acl-base";

  description
    "ACL that contains a mix of entries that
    primarily match on fields in ethernet headers, entries
    that primarily match on fields in IPv6 headers. Matching on
    layer 4 header fields may also exist in the list.";
```

```
}

identity mixed-l2-l3-ipv4-ipv6-acl {
  base "acl:acl-base";

  description
    "ACL that contains a mix of entries that
     primarily match on fields in ethernet headers, entries
     that primarily match on fields in IPv4 headers, and entries
     that primarily match on fields in IPv6 headers. Matching on
     layer 4 header fields may also exist in the list.";
}

identity any-acl {
  base "acl:acl-base";

  description
    "ACL that can contain any pattern to match upon";
}

/*
 * Features
 */
feature l2-acl {
  description
    "Layer 2 ACL supported";
}

feature ipv4-acl {
  description
    "Layer 3 IPv4 ACL supported";
}

feature ipv6-acl {
  description
    "Layer 3 IPv6 ACL supported";
}

feature mixed-ipv4-acl {
  description
    "Layer 2 and Layer 3 IPv4 ACL supported";
}

feature mixed-ipv6-acl {
  description
    "Layer 2 and Layer 3 IPv6 ACL supported";
}
```

```
feature l2-l3-ipv4-ipv6-acl {
  description
    "Layer 2 and any Layer 3 ACL supported.";
}

feature tcp-acl {
  description
    "TCP header ACL supported.";
}

feature udp-acl {
  description
    "UDP header ACL supported.";
}

feature icmp-acl {
  description
    "ICMP header ACL supported.";
}

feature any-acl {
  description
    "ACL for any pattern.";
}

/*
 * Typedefs
 */
typedef acl-type {
  type identityref {
    base acl-base;
  }
  description
    "This type is used to refer to an Access Control List
    (ACL) type";
}

typedef access-control-list-ref {
  type leafref {
    path "/access-lists/acl/acl-name";
  }
  description
    "This type is used by data models that need to reference an
    Access Control List";
}

/*
 * Configuration data nodes
```

```
*/
container access-lists {
  description
    "This is a top level container for Access Control Lists.
    It can have one or more Access Control Lists.";
  list acl {
    key "acl-type acl-name";
    description
      "An Access Control List(ACL) is an ordered list of
      Access List Entries (ACE). Each Access Control Entry has a
      list of match criteria and a list of actions.
      Since there are several kinds of Access Control Lists
      implemented with different attributes for
      different vendors, this
      model accommodates customizing Access Control Lists for
      each kind and for each vendor.";
    leaf acl-name {
      type string;
      description
        "The name of access-list. A device MAY restrict the length
        and value of this name, possibly space and special
        characters are not allowed.";
    }
    leaf acl-type {
      type acl-type;
      description
        "Type of access control list. Indicates the primary intended
        type of match criteria (e.g. ethernet, IPv4, IPv6, mixed,
        etc) used in the list instance.";
    }
  }
  container acl-oper-data {
    config false;
    description
      "Overall Access Control List operational data";
  }
  container access-list-entries {
    description
      "The access-list-entries container contains
      a list of access-list-entries(ACE).";
    list ace {
      key "rule-name";
      ordered-by user;
      description
        "List of access list entries(ACE)";
      leaf rule-name {
        type string;
        description
          "A unique name identifying this Access List
```

```
        Entry(ACE).";
    }

    container matches {
        description
            "The rules in this set determine what fields will be
            matched upon before any action is taken on them.
            The rules are selected based on the feature set
            defined by the server and the acl-type defined.";

        container l2-acl {
            if-feature l2-acl;
            must "../.../acl-type = 'eth-acl'";
            uses packet-fields:acl-eth-header-fields;
            description
                "Rule set for L2 ACL.";
        }

        container ipv4-acl {
            if-feature ipv4-acl;
            must "../.../acl-type = 'ipv4-acl'";
            uses packet-fields:acl-ip-header-fields;
            uses packet-fields:acl-ipv4-header-fields;
            description
                "Rule set that supports IPv4 headers.";
        }

        container ipv6-acl {
            if-feature ipv6-acl;
            must "../.../acl-type = 'ipv6-acl'";
            uses packet-fields:acl-ip-header-fields;
            uses packet-fields:acl-ipv6-header-fields;
            description
                "Rule set that supports IPv6 headers.";
        }

        container l2-l3-ipv4-acl {
            if-feature mixed-ipv4-acl;
            must "../.../acl-type = 'mixed-l2-l3-ipv4-acl'";
            uses packet-fields:acl-eth-header-fields;
            uses packet-fields:acl-ip-header-fields;
            uses packet-fields:acl-ipv4-header-fields;
            description
                "Rule set that is a logical AND (&&) of l2
                and ipv4 headers.";
        }

        container l2-l3-ipv6-acl {
```

```
    if-feature mixed-ipv6-acl;
    must "../../../acl-type = 'mixed-l2-l3-ipv6-acl'";
    uses packet-fields:acl-eth-header-fields;
    uses packet-fields:acl-ip-header-fields;
    uses packet-fields:acl-ipv6-header-fields;
    description
        "Rule set that is a logical AND (&&) of L2
        && IPv6 headers.";
}

container l2-l3-ipv4-ipv6-acl {
    if-feature l2-l3-ipv4-ipv6-acl;
    must "../../../acl-type = 'mixed-l2-l3-ipv4-ipv6-acl'";
    uses packet-fields:acl-eth-header-fields;
    uses packet-fields:acl-ip-header-fields;
    uses packet-fields:acl-ipv4-header-fields;
    uses packet-fields:acl-ipv6-header-fields;
    description
        "Rule set that is a logical AND (&&) of L2
        && IPv4 && IPv6 headers.";
}

container tcp-acl {
    if-feature tcp-acl;
    uses packet-fields:acl-tcp-header-fields;
    description
        "Rule set that defines TCP headers.";
}

container udp-acl {
    if-feature udp-acl;
    uses packet-fields:acl-udp-header-fields;
    description
        "Rule set that defines UDP headers.";
}

container icmp-acl {
    if-feature icmp-acl;
    uses packet-fields:acl-icmp-header-fields;
    description
        "Rule set that defines ICMP headers.";
}

container any-acl {
    if-feature any-acl;
    must "../../../acl-type = 'any-acl'";
    presence "Matches any";
    description
```



```

        "Rule set that allows for a any ACL.";
    }
}

container actions {
    description
        "Definitions of action criteria for this Access List
        Entry.";
    choice packet-handling {
        default "deny";
        description
            "Packet handling action.";
        case deny {
            leaf deny {
                type empty;
                description
                    "Deny action.";
            }
        }
        case permit {
            leaf permit {
                type empty;
                description
                    "Permit action.";
            }
        }
    }
}

leaf logging {
    type boolean;
    description
        "Log the rule on which the match occurred.
        Setting the value to true enables logging,
        whereas setting the value to false disables it.";
}

}
/*
 * Operational state data nodes
 */
container ace-oper-data {
    config false;
    description
        "Operational data for this Access List Entry.";
    leaf match-counter {
        type yang:counter64;
        description
            "Number of matches for this Access List Entry";
    }
}
}

```

```

    }
  }
}

```

<CODE ENDS>

#### 4.2. IETF Packet Fields module

The packet fields module defines the necessary groups for matching on fields in the packet including ethernet, ipv4, ipv6, and transport layer fields. The 'acl-type' node determines which of these fields get included for any given ACL with the exception of TCP, UDP and ICMP header fields. Those fields can be used in conjunction with any of the above layer 2 or layer 3 fields.

Since the number of match criteria is very large, the base draft does not include these directly but references them by "uses" to keep the base module simple. In case more match conditions are needed, those can be added by augmenting choices within container "matches" in ietf-access-control-list.yang model.

<CODE BEGINS> file "ietf-packet-fields@2017-06-16.yang"

```

module ietf-packet-fields {
  namespace "urn:ietf:params:xml:ns:yang:ietf-packet-fields";
  prefix packet-fields;

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working
    Group";

  contact
    "WG Web: http://tools.ietf.org/wg/netmod/
    WG List: netmod@ietf.org

    Editor: Dean Bogdanovic
    ivandean@gmail.com
    Editor: Mahesh Jethanandani

```

maresh@cisco.com  
Editor: Lisa Huang  
lyihuang16@gmail.com  
Editor: Sonal Agarwal  
agarwaso@cisco.com  
Editor: Dana Blair  
dblair@cisco.com";

description

"This YANG module defines groupings that are used by ietf-access-control-list YANG module. Their usage is not limited to ietf-access-control-list and can be used anywhere as applicable.

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. 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>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

revision 2017-06-16 {

description

"Added header fields for TCP, UDP, and ICMP.";

reference

"RFC XXX: Network Access Control List (ACL) YANG Data Model.";

}

revision 2016-10-12 {

description

"Initial version of packet fields used by ietf-access-control-list";

reference

"RFC XXXX: Network Access Control List (ACL) YANG Data Model";

}

grouping acl-transport-header-fields {

description

"Transport header fields";

container source-port-range {

presence "Enables setting source port range";

description

"Inclusive range representing source ports to be used.

```
        When only lower-port is present, it represents a single port.";
    leaf lower-port {
        type inet:port-number;
        mandatory true;
        description
            "Lower boundary for port.";
    }
    leaf upper-port {
        type inet:port-number;
        must ". >= ../lower-port" {
            error-message
                "The upper-port must be greater than or equal
                 to lower-port";
        }
        description
            "Upper boundary for port . If existing, the upper port
             must be greater or equal to lower-port.";
    }
}

container destination-port-range {
    presence "Enables setting destination port range";
    description
        "Inclusive range representing destination ports to be used.
         When only lower-port is present, it represents a single
         port.";

    leaf lower-port {
        type inet:port-number;
        mandatory true;
        description

            "Lower boundary for port.";
    }
    leaf upper-port {
        type inet:port-number;
        must ". >= ../lower-port" {
            error-message
                "The upper-port must be greater than or equal
                 to lower-port";
        }
    }

    description
        "Upper boundary for port. If existing, the upper port must
         be greater or equal to lower-port";
    }
}
}
```

```
grouping acl-ip-header-fields {
  description
    "IP header fields common to ipv4 and ipv6";
  reference
    "RFC 791.";

  leaf tos {
    type uint8;
    description
      "Also known as Traffic Class in IPv6. The Type of Service (TOS)
       provides an indication of the abstract parameters of the
       quality of service desired.";
    reference
      "RFC 719, RFC 2460";
  }

  leaf length {
    type uint16;
    description
      "In IPv4 header field, this field is known as the Total Length.
       Total Length is the length of the datagram, measured in octets,
       including internet header and data.

       In IPv6 header field, this field is known as the Payload
       Length, the length of the IPv6 payload, i.e. the rest of
       the packet following the IPv6 header, in octets.";
    reference
      "RFC 719, RFC 2460";
  }

  leaf ttl {
    type uint8;
    description
      "This field indicates the maximum time the datagram is allowed
       to remain in the internet system. If this field contains the
       value zero, then the datagram must be destroyed.

       In IPv6, this field is known as the Hop Limit.";
    reference "RFC 719, RFC 2460";
  }

  leaf protocol {
    type uint8;
    description
      "Internet Protocol number.";
  }
  uses acl-transport-header-fields;
}
```

```
grouping acl-ipv4-header-fields {
  description
    "Fields in IPv4 header.";

  leaf ihl {
    type uint8 {
      range "5..60";
    }
    description
      "An IPv4 header field, the Internet Header Length (IHL) is
       the length of the internet header in 32 bit words, and
       thus points to the beginning of the data. Note that the
       minimum value for a correct header is 5.";
  }

  leaf flags {
    type bits {
      bit reserved {
        position 0;
        description
          "Reserved. Must be zero.";
      }
      bit fragment {
        position 1;
        description
          "Setting value to 0 indicates may fragment, while setting
           the value to 1 indicates do not fragment.";
      }
      bit more {
        position 2;
        description
          "Setting the value to 0 indicates this is the last fragment,
           and setting the value to 1 indicates more fragments are
           coming.";
      }
    }
    description
      "Bit definitions for the flags field in IPv4 header.";
  }

  leaf offset {
    type uint16 {
      range "20..65535";
    }
    description
      "The fragment offset is measured in units of 8 octets (64 bits).
       The first fragment has offset zero. The length is 13 bits";
  }
}
```

```
leaf identification {
  type uint16;
  description
    "An identifying value assigned by the sender to aid in
    assembling the fragments of a datagram.";
}

leaf destination-ipv4-network {
  type inet:ipv4-prefix;
  description
    "Destination IPv4 address prefix.";
}
leaf source-ipv4-network {
  type inet:ipv4-prefix;
  description
    "Source IPv4 address prefix.";
}
}

grouping acl-ipv6-header-fields {
  description
    "Fields in IPv6 header";

  leaf next-header {
    type uint8;
    description
      "Identifies the type of header immediately following the
      IPv6 header. Uses the same values as the IPv4 Protocol
      field.";
    reference
      "RFC 2460";
  }

  leaf destination-ipv6-network {
    type inet:ipv6-prefix;
    description
      "Destination IPv6 address prefix.";
  }

  leaf source-ipv6-network {
    type inet:ipv6-prefix;
    description
      "Source IPv6 address prefix.";
  }

  leaf flow-label {
    type inet:ipv6-flow-label;
    description
```

```
        "IPv6 Flow label.";
    }
    reference
        "RFC 4291: IP Version 6 Addressing Architecture
        RFC 4007: IPv6 Scoped Address Architecture
        RFC 5952: A Recommendation for IPv6 Address Text
        Representation";
}

grouping acl-eth-header-fields {
    description
        "Fields in Ethernet header.";

    leaf destination-mac-address {
        type yang:mac-address;
        description
            "Destination IEEE 802 MAC address.";
    }
    leaf destination-mac-address-mask {
        type yang:mac-address;
        description
            "Destination IEEE 802 MAC address mask.";
    }
    leaf source-mac-address {
        type yang:mac-address;
        description
            "Source IEEE 802 MAC address.";
    }
    leaf source-mac-address-mask {
        type yang:mac-address;
        description
            "Source IEEE 802 MAC address mask.";
    }
    leaf ether-type {
        type string {
            pattern '[0-9a-fA-F]{4}';
        }
        description
            "The Ethernet Type (or Length) value represented
            in the canonical order defined by IEEE 802.
            The canonical representation uses lowercase
            characters.

            Note: This is not the most ideal way to define
            ether-types. Ether-types are well known types
            and are registered with RAC in IEEE. So they
            should well defined types with values. For now
            this model is defining it as a string."
```



```
        There is a note out to IEEE that needs to be
        turned into a liaison statement asking them to
        define all ether-types for the industry to use.";
    reference
        "IEEE 802-2014 Clause 9.2";
}
reference
    "IEEE 802: IEEE Standard for Local and Metropolitan
    Area Networks: Overview and Architecture.";
}

grouping acl-tcp-header-fields {
    description
        "Collection of TCP header fields that can be used to
        setup a match filter.";

    leaf sequence-number {
        type uint32;
        description
            "Sequence number that appears in the packet.";
    }

    leaf acknowledgement-number {
        type uint32;
        description
            "The acknowledgement number that appears in the
            packet.";
    }

    leaf data-offset {
        type uint8 {
            range "5..15";
        }
        description
            "Specifies the size of the TCP header in 32-bit
            words. The minimum size header is 5 words and
            the maximum is 15 words thus giving the minimum
            size of 20 bytes and maximum of 60 bytes,
            allowing for up to 40 bytes of options in the
            header.";
    }

    leaf reserved {
        type uint8;
        description
            "Reserved for future use.";
    }
}
```

```
leaf flags {
  type uint16;
  description
    "Also known as Control Bits. Contains 9 1-bit flags.";
}

leaf window-size {
  type uint16;
  description
    "The size of the receive window, which specifies
    the number of window size units (by default,
    bytes) (beyond the segment identified by the
    sequence number in the acknowledgment field)
    that the sender of this segment is currently
    willing to receive.";
}

leaf urgent-pointer {
  type uint16;
  description
    "This field s an offset from the sequence number
    indicating the last urgent data byte.";
}

leaf options {
  type uint32;
  description
    "The length of this field is determined by the
    data offset field. Options have up to three
    fields: Option-Kind (1 byte), Option-Length
    (1 byte), Option-Data (variable). The Option-Kind
    field indicates the type of option, and is the
    only field that is not optional. Depending on
    what kind of option we are dealing with,
    the next two fields may be set: the Option-Length
    field indicates the total length of the option,
    and the Option-Data field contains the value of
    the option, if applicable.";
}

}

grouping acl-udp-header-fields {
  description
    "Collection of UDP header fields that can be used
    to setup a match filter.";

  leaf length {
    type uint16;
  }
}
```

```
    description
      "A field that specifies the length in bytes of
       the UDP header and UDP data. The minimum
       length is 8 bytes because that is the length of
       the header. The field size sets a theoretical
       limit of 65,535 bytes (8 byte header + 65,527
       bytes of data) for a UDP datagram. However the
       actual limit for the data length, which is
       imposed by the underlying IPv4 protocol, is
       65,507 bytes (65,535 minus 8 byte UDP header
       minus 20 byte IP header).

       In IPv6 jumbograms it is possible to have
       UDP packets of size greater than 65,535 bytes.
       RFC 2675 specifies that the length field is set
       to zero if the length of the UDP header plus
       UDP data is greater than 65,535."
    }
  }
  grouping acl-icmp-header-fields {
    description
      "Collection of ICMP header fields that can be
       used to setup a match filter.";

    leaf type {
      type uint8;
      description
        "Also known as Control messages.";
      reference "RFC 792";
    }

    leaf code {
      type uint8;
      description
        "ICMP subtype. Also known as Control messages.";
    }

    leaf rest-of-header {
      type uint32;
      description
        "Four-bytes field, contents vary based on the
         ICMP type and code.";
    }
  }
}
```

<CODE ENDS>

#### 4.3. An ACL Example

Requirement: Deny tcp traffic from 10.10.10.1/24, destined to 11.11.11.1/24.

Here is the acl configuration xml for this Access Control List:

```
<?xml version='1.0' encoding='UTF-8'?>
  <data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <access-lists xmlns="urn:ietf:params:xml:ns:yang:
      ietf-access-control-list">
      <acl>
        <acl-name>sample-ipv4-acl</acl-name>
        <acl-type>ipv4</acl-type>
        <access-list-entries>
          <ace>
            <rule-name>rule1</rule-name>
            <matches>
              <source-ipv4-network>
                10.10.10.1/24
              </source-ipv4-network>
              <destination-ipv4-network>
                11.11.11.1/24
              </destination-ipv4-network>
            </matches>
            <actions>
              <deny />
            </actions>
            <protocol>
              tcp
            </protocol>
          </ace>
        </access-list-entries>
      </acl>
    </access-lists>
  </data>
```

The acl and aces can be described in CLI as the following:

```
access-list ipv4 sample-ipv4-acl
deny tcp 10.10.10.1/24 11.11.11.1/24
```

#### 4.4. Port Range Usage Example

When a lower-port and an upper-port are both present, it represents a range between lower-port and upper-port with both the lower-port and upper-port are included. When only a lower-port presents, it represents a single port.

With the follow XML snippet:

```
<source-port-range>
  <lower-port>16384</lower-port>
  <upper-port>16387</upper-port>
</source-port-range>
```

This represents source ports 16384,16385, 16386, and 16387.

With the follow XML snippet:

```
<source-port-range>
  <lower-port>16384</lower-port>
  <upper-port>65535</upper-port>
</source-port-range>
```

This represents source ports greater than/equal to 16384 and less than equal to 65535.

With the follow XML snippet:

```
<source-port-range>
  <lower-port>21</lower-port>
</source-port-range>
```

This represents port 21.

## 5. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The NETCONF Access Control Model ( NACM [RFC6536]) provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of data nodes defined in the YANG module which are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., <edit-config>) to these data nodes without proper protection can have a negative effect on network operations.

These are the subtrees and data nodes and their sensitivity/vulnerability:

/access-lists/acl/access-list-entries: This list specifies all the configured access list entries on the device. Unauthorized write access to this list can allow intruders to access and control the system. Unauthorized read access to this list can allow intruders to spoof packets with authorized addresses thereby compromising the system.

## 6. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made:

URI: urn:ietf:params:xml:ns:yang:ietf-access-control-list

URI: urn:ietf:params:xml:ns:yang:ietf-packet-fields

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

name: ietf-access-control-list namespace:  
urn:ietf:params:xml:ns:yang:ietf-access-control-list prefix: ietf-acl  
reference: RFC XXXX

name: ietf-packet-fields namespace: urn:ietf:params:xml:ns:yang:ietf-packet-fields prefix: ietf-packet-fields reference: RFC XXXX

## 7. Acknowledgements

Alex Clemm, Andy Bierman and Lisa Huang started it by sketching out an initial IETF draft in several past IETF meetings. That draft included an ACL YANG model structure and a rich set of match filters, and acknowledged contributions by Louis Fourie, Dana Blair, Tula Kraiser, Patrick Gili, George Serpa, Martin Bjorklund, Kent Watsen, and Phil Shafer. Many people have reviewed the various earlier drafts that made the draft went into IETF charter.

Dean Bogdanovic, Kiran Agrahara Sreenivasa, Lisa Huang, and Dana Blair each evaluated the YANG model in previous draft separately and then work together, to created a new ACL draft that can be supported by different vendors. The new draft removes vendor specific features, and gives examples to allow vendors to extend in their own proprietary ACL. The earlier draft was superseded with the new one that received more participation from many vendors.

Authors would like to thank Jason Sterne, Lada Lhotka, Juergen Schoenwalder, and David Bannister for their review of and suggestions to the draft.

## 8. Open Issues

- o The current model does not support the concept of "containers" used to contain multiple addresses per rule entry.
- o The current model defines 'any' rule as a presence container, allowing a user to define any 'any' rule.
- o The model defines 'ether-type' node as a string. Ideally, this should be a well defined list of all Ethernet Types assigned by IEEE.
- o Should this draft include route-policy definition as defined in draft-ietf-rtgwg-policy-model?

## 9. References

### 9.1. Normative References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.

## 9.2. Informative References

- [I-D.ietf-netmod-yang-tree-diagrams]  
 Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-00 (work in progress), June 2017.
- [RFC5101] Claise, B., Ed., "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information", RFC 5101, DOI 10.17487/RFC5101, January 2008, <<http://www.rfc-editor.org/info/rfc5101>>.

## Appendix A. Extending ACL model examples

### A.1. Example of extending existing model for route filtering

With proposed modular design, it is easy to extend the model with other features. Those features can be standard features, like route filters. Route filters match on specific IP addresses or ranges of prefixes. Much like ACLs, they include some match criteria and corresponding match action(s). For that reason, it is very simple to extend existing ACL model with route filtering. The combination of a route prefix and prefix length along with the type of match determines how route filters are evaluated against incoming routes. Different vendors have different match types and in this model we are using only ones that are common across all vendors participating in this draft. As in this example, the base ACL model can be extended with company proprietary extensions, described in the next section.

```
module: example-ext-route-filter
augment /ietf-acl:access-lists/ietf-acl:acl/
ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:matches:
  +--rw (route-prefix)?
    +--:(range)
      +--rw (ipv4-range)?
        | +--:(v4-lower-bound)
        | | +--rw v4-lower-bound?   inet:ipv4-prefix
        | +--:(v4-upper-bound)
        | | +--rw v4-upper-bound?   inet:ipv4-prefix
      +--rw (ipv6-range)?
        | +--:(v6-lower-bound)
        | | +--rw v6-lower-bound?   inet:ipv6-prefix
        | +--:(v6-upper-bound)
        | | +--rw v6-upper-bound?   inet:ipv6-prefix

file "example-ext-route-filter@2016-10-12.yang"
module example-ext-route-filter {
  namespace "urn:ietf:params:xml:ns:yang:example-ext-route-filter";
```



```
prefix example-ext-route-filter;
import ietf-inet-types {
  prefix "inet";
}
import ietf-access-control-list {
  prefix "ietf-acl";
}

organization
  "Route model group.";

contact
  "abc@abc.com";

description "
  This module describes route filter as a collection of
  match prefixes. When specifying a match prefix, you
  can specify an exact match with a particular route or
  a less precise match. You can configure either a
  common action that applies to the entire list or an
  action associated with each prefix.
  ";
revision 2016-10-12 {
  description
    "Creating Route-Filter extension model based on
    ietf-access-control-list model";
  reference " ";
}
augment "/ietf-acl:access-lists/ietf-acl:acl/"
+ "ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:matches"{
  description "
    This module augments the matches container in the ietf-acl
    module with route filter specific actions
    ";
  choice route-prefix{
    description "Define route filter match criteria";
    case range {
      description
        "Route falls between the lower prefix/prefix-length
        and the upperprefix/prefix-length.";
      choice ipv4-range {
        description "Defines the IPv4 prefix range";
        leaf v4-lower-bound {
          type inet:ipv4-prefix;
          description
            "Defines the lower IPv4 prefix/prefix length";
        }
        leaf v4-upper-bound {
```

```

        type inet:ipv4-prefix;
        description
            "Defines the upper IPv4 prefix/prefix length";
    }
}
choice ipv6-range {
    description "Defines the IPv6 prefix/prefix range";
    leaf v6-lower-bound {
        type inet:ipv6-prefix;
        description
            "Defines the lower IPv6 prefix/prefix length";
    }
    leaf v6-upper-bound {
        type inet:ipv6-prefix;
        description
            "Defines the upper IPv6 prefix/prefix length";
    }
}
}
}
}
}
}
}
}

```

#### A.2. A company proprietary module example

Access control list typically does not exist in isolation. Instead, they are associated with a certain scope in which they are applied, for example, an interface of a set of interfaces. How to attach an access control list to an interface (or other system artifact) is outside the scope of this model, as it depends on the specifics of the system model that is being applied. However, in general, the general design pattern will involve adding a data node with a reference, or set of references, to ACLs that are to be applied to the interface. For this purpose, the type definition "access-control-list-ref" can be used.

Module "example-newco-acl" is an example of company proprietary model that augments "ietf-acl" module. It shows how to use 'augment' with an XPath expression to add additional match criteria, action criteria, and default actions when no ACE matches found, as well how to attach an Access Control List to an interface. All these are company proprietary extensions or system feature extensions. "example-newco-acl" is just an example and it is expected from vendors to create their own proprietary models.

The following figure is the tree structure of example-newco-acl. In this example, /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:access-

list-entries/ ietf-acl:ace/ietf-acl:matches are augmented with two new choices, protocol-payload-choice and metadata. The protocol-payload-choice uses a grouping with an enumeration of all supported protocol values. Metadata matches apply to fields associated with the packet but not in the packet header such as input interface or overall packet length. In other example, /ietf-acl:access-lists/ ietf-acl:acl/ietf-acl:access-list-entries/ ietf-acl:ace/ietf-acl:actions are augmented with new choice of actions.

```

module: example-newco-acl
augment /ietf-acl:access-lists/ietf-acl:acl/
ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:matches:
  +--rw vlan-tagged?      uint16
  +--rw mpls-unicast?     uint16
  +--rw mpls-multicast?   uint16
  +--rw ipv4?             uint16
  +--rw ipv6?             uint16
augment /ietf-acl:access-lists/ietf-acl:acl/
ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:matches:
  +--rw ipv4-ttl?         uint8
  +--rw ipv4-len?         uint16
  +--rw ipv4-ihl?         uint8
  +--rw ipv4-id?          uint16
  +--rw ipv4-flags?       ipv4-flags-type
  +--rw ipv4-offset?      uint16
augment /ietf-acl:access-lists/ietf-acl:acl/
ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:matches:
  +--rw (protocol-payload-choice)?
  |   +---:(protocol-payload)
  |       +--rw protocol-payload* [value-keyword]
  |       +--rw value-keyword      enumeration
  +--rw (metadata)?
  |   +---:(interface-name)
  |       +--rw interface-name* [input-interface]
  |       +--rw input-interface   ietf-if:interface-ref
augment /ietf-acl:access-lists/ietf-acl:acl/
ietf-acl:access-list-entries/ietf-acl:ace/ietf-acl:actions:
  +--rw (action)?
  |   +---:(count)
  |       +--rw count?          string
  |   +---:(policer)
  |       +--rw policer?        string
  |   +---:(hiearchical-policer)
  |       +--rw hierarchitacl-policer? string
augment /ietf-acl:access-lists/ietf-acl:acl/
  +--rw default-actions
  +--rw deny?      empty
augment /ietf-if:interfaces/ietf-if:interface:

```

```

    +--rw acl
      +--rw acl-name?          ietf-acl:access-control-list-ref
      +--ro match-counter?     yang:counter64
      +--rw (direction)?
        +--:(in)
          | +--rw in?          empty
          +--:(out)
            +--rw out?         empty
augment /ietf-acl:access-lists/ietf-acl:acl/ietf-acl:acl-oper-data:
  +--ro targets
    +--ro (interface)?
      +--:(interface-name)
        +--ro interface-name*  ietf-if:interface-ref

module example-newco-acl {

  yang-version 1.1;

  namespace "urn:newco:params:xml:ns:yang:example-newco-acl";

  prefix example-newco-acl;

  import ietf-access-control-list {
    prefix "ietf-acl";
  }

  import ietf-interfaces {
    prefix "ietf-if";
  }

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "Newco model group.";

  contact
    "abc@newco.com";
  description
    "This YANG module augment IETF ACL Yang.";

  revision 2016-10-12{
    description
      "Creating NewCo proprietary extensions to ietf-acl model";
    reference
      "RFC XXXX: Network Access Control List (ACL)
       YANG Data Model";
  }
}
```

```
}

typedef known-ether-type {
  type enumeration {
    enum "ipv4" {
      value 2048; // 0x0800
      description "Internet Protocol version 4 (IPv4)";
    }
    enum "vlan-tagged" {
      value 33024; // 0x8100
      description
        "VLAN-tagged frame (IEEE 802.1Q) & Shortest Path
        Bridging IEEE 802.1aq[4]";
    }
    enum "ipv6" {
      value 34525; // 0x86DD
      description "Internet Protocol Version 6 (IPv6)";
    }
    enum "mpls-unicast" {
      value 34887; // 0x8847
      description "MPLS unicast";
    }
    enum "mpls-multicast" {
      value 34888; // 0x8848
      description "MPLS multicast";
    }
  }
  description "Listing supported Ethertypes";
}

typedef ipv4-flags-type {
  type bits {
    bit ipv4-reserved {
      position 0;
      description "reserved bit";
    }
    bit ipv4-DF {
      position 1;
      description "DF bit";
    }
    bit ipv4-MF {
      position 2;
      description "MF bit";
    }
  }
  description "IPv4 flag types";
}
```

```
augment "/ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:access-list-entries/ietf-acl:ace/" +
    "ietf-acl:matches" {
when "ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:acl-type = 'ace-eth'";

description "additional MAC header matching";

    leaf vlan-tagged {
        type uint16;
        description "Ethernet frame with VLAN tag";
    }

    leaf mpls-unicast {
        type uint16;
        description "Ethernet frame with MPLS unicast payload";
    }

    leaf mpls-multicast {
        type uint16;
        description "Ethernet frame with MPLS multicast payload";
    }

    leaf ipv4 {
        type uint16;
        description "Ethernet frame with IPv4 unicast payload";
    }

    leaf ipv6 {
        type uint16;
        description "Ethernet frame with IPv4 unicast payload";
    }
}
augment "/ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:access-list-entries/ietf-acl:ace/" +
    "ietf-acl:matches" {
when "ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:acl-type = 'ipv4-acl'";

description "additional IP header information";

    leaf ipv4-ttl {
        type uint8;
        description "time to live of a given packet as
            defined in RFC791";
    }

    leaf ipv4-len {
```

```
    type uint16;
    description "total packet length as defined in RFC791";
}

leaf ipv4-ihl {
    type uint8 {
        range 0..15;
    }
    description "Internet Header Length in 32 bit words
        (see RFC791). Note that while the minimum
        value for this field in a packet is 5,
        we leave open the possibility here that
        the packet has been corrupted.";
}

leaf ipv4-id {
    type uint16;
    description "Identification as described in RFC791";
}

leaf ipv4-flags {
    type ipv4-flags-type;
    description "IPv4 flags as defined in RFC791";
}

leaf ipv4-offset {
    type uint16 {
        range 0..8191;
    }
    description "Matches on the packet fragment offset";
}

augment "/ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:access-list-entries/ietf-acl:ace/" +
    "ietf-acl:matches" {
    description "Newco proprietary simple filter matches";
    choice protocol-payload-choice {
        description "Newco proprietary payload match condition";
        list protocol-payload {
            key value-keyword;
            ordered-by user;
            description "Match protocol payload";
            uses match-simple-payload-protocol-value;
        }
    }

    choice metadata {
```

```
        description "Newco proprietary interface match condition";
        list interface-name {
            key input-interface;
            ordered-by user;
            description "Match interface name";
            uses metadata;
        }
    }
}

augment "/ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:access-list-entries/ietf-acl:ace/" +
    "ietf-acl:actions" {
    description "Newco proprietary simple filter actions";
    choice action {
        description "";
        case count {
            description "Count the packet in the named counter";
            leaf count {
                type string;
                description "";
            }
        }
        case policer {
            description "Name of policer to use to rate-limit traffic";
            leaf policer {
                type string;
                description "";
            }
        }
        case hierarchical-policer {
            description "Name of hierarchical policer to use to
            rate-limit traffic";
            leaf hierarchitacl-policer {
                type string;
                description "";
            }
        }
    }
}

augment "/ietf-acl:access-lists/ietf-acl:acl" {
    description "Newco proprietary default action";
    container default-actions {
        description
            "Actions that occur if no access-list entry is matched.";
        leaf deny {
            type empty;
        }
    }
}
```



```
        description "";
    }
}

grouping metadata {
    description
        "Fields associated with a packet which are not in
        the header.";
    leaf input-interface {
        type ietf-if:interface-ref {
            require-instance false;
        }
        description
            "Packet was received on this interface";
    }
}

grouping match-simple-payload-protocol-value {
    description "Newco proprietary payload";
    leaf value-keyword {
        type enumeration {
            enum icmp {
                description "Internet Control Message Protocol";
            }
            enum icmp6 {
                description "Internet Control Message Protocol Version 6";
            }
            enum range {
                description "Range of values";
            }
        }
    }
    description "(null)";
}

augment "/ietf-if:interfaces/ietf-if:interface" {
    description "Apply ACL to interfaces";
    container acl {
        description "ACL related properties.";
        leaf acl-name {
            type ietf-acl:access-control-list-ref;
            description "Access Control List name.";
        }
        leaf match-counter {
            type yang:counter64;
            config false;
        }
    }
}
```

```

        description
            "Total match count for Access Control
            List on this interface";
    }
    choice direction {
        description "Applying ACL in which traffic direction";
        leaf in {
            type empty;
            description "Inbound traffic";
        }
        leaf out {
            type empty;
            description "Outbound traffic";
        }
    }
}

augment "/ietf-acl:access-lists/ietf-acl:acl/" +
    "ietf-acl:acl-oper-data" {
    description
        "This is an example on how to apply acl to a target to collect
        operational data";
    container targets {
        description "To which object is the ACL attached to";
        choice interface {
            description
                "Access Control List was attached to this interface";
            leaf-list interface-name {
                type ietf-if:interface-ref {
                    require-instance true;
                }
            }
            description "Attached to this interface name";
        }
    }
}
}

```

Draft authors expect that different vendors will provide their own yang models as in the example above, which is the augmentation of the base model

### A.3. Example to augment model with mixed ACL type

As vendors (or IETF) add more features to ACL, the model is easily augmented. One of such augmentations can be to add support for mixed type of ACLs, where `acl-type-base` can be augmented like in example below:

```
identity mixed-l3-acl {
  base "access-control-list:acl-type-base";
  description "ACL that contains a mix of entries that
    primarily match on fields in IPv4 headers and entries
    that primarily match on fields in IPv6 headers.
    Matching on layer 4 header fields may also exist in the
    list. An acl of type mixed-l3-acl does not contain
    matches on fields in the ethernet header.";
}

identity mixed-l2-l3-acl {
  base "access-control-list:acl-type-base";
  description "ACL that contains a mix of entries that
    primarily match on fields in ethernet headers, entries
    that primarily match on fields in IPv4 headers,
    and entries that primarily match on fields in IPv6
    headers. Matching on layer 4 header fields may also
    exist in the list.";
}
```

#### A.4. Linux nftables

As Linux platform is becoming more popular as networking platform, the Linux data model is changing. Previously ACLs in Linux were highly protocol specific and different utilities were used (iptables, ip6tables, arptables, ebtables), so each one had separate data model. Recently, this has changed and a single utility, nftables, has been developed. With a single application, it has a single data model for firewall filters and it follows very similarly to the ietf-access-control list module proposed in this draft. The nftables support input and output ACEs and each ACE can be defined with match and action.

The example in Section 4.3 can be configured using nftable tool as below.

```
nft add table ip filter
nft add chain filter input
nft add rule ip filter input ip protocol tcp ip saddr \
  10.10.10.1/24 drop
```

The configuration entries added in nftable would be.

```
table ip filter {  
  chain input {  
    ip protocol tcp ip saddr 10.10.10.1/24 drop  
  }  
}
```

We can see that there are many similarities between Linux nftables and IETF ACL YANG data models and its extension models. It should be fairly easy to do translation between ACL YANG model described in this draft and Linux nftables.

#### Authors' Addresses

Dean Bogdanovic  
Volta Networks

Email: [ivandean@gmail.com](mailto:ivandean@gmail.com)

Mahesh Jethanandani  
Cisco Systems, Inc

Email: [mjethanandani@gmail.com](mailto:mjethanandani@gmail.com)

Lisa Huang  
General Electric

Email: [lyihuang16@gmail.com](mailto:lyihuang16@gmail.com)

Sonal Agarwal  
Cisco Systems, Inc.

Email: [agarwaso@cisco.com](mailto:agarwaso@cisco.com)

Dana  
Cisco Systems, INC

Email: [dblair@cisco.com](mailto:dblair@cisco.com)