Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 8, 2017

R. Wilton, Ed.
D. Ball
T. Singh
Cisco Systems
S. Sivaraj
Juniper Networks
July 7, 2016

Common Interface Extension YANG Data Models
draft-ietf-netmod-intf-ext-yang-01

Abstract

This document defines two YANG modules that augment the Interfaces data model defined in the "YANG Data Model for Interface Management" with additional configuration and operational data nodes to support common lower layer interface properties, such as interface MTU. These properties are common to many types of interfaces on network routers and switches and are implemented by multiple network equipment vendors with similar semantics, even though some of the features are not formally defined in any published standard.

Table of Contents

1. Introduction

This document defines two YANG RFC 6020 [RFC6020] modules for the management of network interfaces. It defines various augmentations to the generic interfaces data model defined in RFC 7223 [RFC7223] to support configuration of lower layer interface properties that are common across many types of network interface.

One of the aims of this draft is to provide a standard namespace and path for these configuration items regardless of the underlying interface type. For example a standard namespace and path for configuring or reading the MAC address associated with an interface is provided that can be used for any interface type that uses Ethernet framing.

Several of the augmentations defined here are not backed by any formal standard specification. Instead, they are for features that are commonly implemented in equivalent ways by multiple independent network equipment vendors. The aim of this draft is to define common paths and leaves for the configuration of these equivalent features in a uniform way, making it easier for users of the YANG model to access these features in a vendor independent way. Where necessary, a description of the expected behavior is also provided with the aim of ensuring vendors implementations are consistent with the specified behaviour.

Given that the modules contain a collection of discrete features with the common theme that they generically apply to interfaces, it is plausible that not all implementors of the YANG module will decide to support all features. Hence separate feature keywords are defined for each logically discrete feature to allow implementors the flexibility to choose which specific parts of the model they support.

The augmentations are split into two separate YANG modules that each focus on a particular area of functionality. The two YANG modules defined in this internet draft are:

interface-extensions.yang - Defines extensions to the IETF interface data model to support common configuration data nodes.

etherlike-interfaces.yang - Defines a module for any configuration and operational data nodes that are common across interfaces that use Ethernet framing.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.

- o Abbreviations before data node names: "rw" means configuration (read-write), and "ro" means state data (read-only).

- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list or leaf-list.

- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Objectives

The aim of of the YANG modules contained in this draft is to provide standard definitions for common interface based configuration on network devices.

The expectation is that the YANG leaves that are being defined are fairly widely implemented by network vendors. However, the features described here are mostly not backed by formal standards because they are fairly basic in their behavior and do not need to interoperate with other devices. Where required a concise explanation of the expected behavior is also provided to ensure consistency between vendors.

3. Interfaces Extensions Module

The Interfaces Common module provides some basic extensions to the IETF interfaces YANG module.

The module provides:

- o A bandwidth configuration leaf to specify the bandwidth available on an interface to control routing metrics.

- o A carrier delay feature used to provide control over short lived link state flaps.

- o An interface link state dampening feature that is used to provide control over longer lived link state flaps.

- o An encapsulation container and extensible choice statement for use by any interface types that allow for configurable L2 encapsulations.

o A loopback configuration leaf that is primarily aimed at loopback at the physical layer.

o MTU configuration leaves applicable to all packet/frame based interfaces.

o A transport layer leaf to indicate whether the interface handles traffic at L1, L2 or L3.

o A parent interface leaf useable for all types of sub-interface that are children of parent interfaces.

The "interface-extensions" YANG module has the following structure:

```
module: ietf-interfaces-common
augment /if:interfaces/if:interface:
   +--rw bandwidth?   uint64
augment /if:interfaces/if:interface:
   +--rw carrier-delay
      +--rw down?   uint32
      +--rw up?     uint32
augment /if:interfaces/if:interface:
   +--rw dampening!
      +--rw half-life?           uint32
      +--rw reuse?               uint32
      +--rw suppress?            uint32
      +--rw max-suppress-time?   uint32
augment /if:interfaces/if:interface:
   +--rw encapsulation
      +--rw (encaps-type)?
augment /if:interfaces/if:interface:
   +--rw loopback?   identityref
augment /if:interfaces/if:interface:
   +--rw l2-mtu?   uint16 {configurable-l2-mtu}?
augment /if:interfaces/if:interface:
   +--rw parent-interface?   if:interface-ref
augment /if:interfaces/if:interface:
   +--rw transport-layer?   enumeration
```

3.1. Bandwidth

The bandwidth configuration leaf allows the specified bandwidth of an interface to be reduced from the inherent interface bandwidth. The bandwidth leaf affects the routing metric cost associated with the interface.

Note that the bandwidth leaf does not actually limit the amount of traffic that can be sent/received over the interface. If required,

interface traffic can be limited to the required bandwidth by configuring an explicit QoS policy.

Note for reviewers: Given that the bandwidth only controls routing metrics, it may be more appropriate for this leaf, or an equivalent, to be defined as part of one of the routing YANG modules. Although conversely, it is also worth considering that the corresponding existing CLI configuration command is an interface level bandwidth command in many implementations.

3.2. Carrier Delay

The carrier delay feature augments the IETF interfaces data model with configuration for a simple algorithm that is used, generally on physical interfaces, to suppress short transient changes in the interface link state. It can be used in conjunction with the dampening feature described in Section 3.3 to provide effective control of unstable links and unwanted state transitions.

The principal of the carrier delay feature is to use a short per interface timer to ensure that any interface link state transition that occurs and reverts back within the specified time interval is entirely suppressed without providing any signalling to any upper layer protocols that the state transition has occurred. E.g. in the case that the link state transition is suppressed then there is no change of the /if:interfaces-state/if:interface/oper-status or /if:interfaces-state/if:interfaces/last-change leaves for the interface that the feature is operating on. One obvious side effect of using this feature that is worth noting is that any state transition will always be delayed by the specified time interval.

The configuration allows for separate timer values to be used in the suppression of down->up->down link transitions vs up->down->up link transitions.

The carrier delay down timer leaf specifies the amount of time that an interface that is currently in link up state must be continuously down before the down state change is reported to higher level protocols. Use of this timer can cause traffic to be black holed for the configured value and delay reconvergence after link failures, therefore its use is normally restricted to cases where it is necessary to allow enough time for another protection mechanism (such as an optical layer automatic protection system) to take effect.

The carrier delay up timer leaf specifies the amount of time that an interface that is currently in link down state must be continuously up before the down->up link state transition is reported to higher level protocols. This timer is generally useful as a debounce

mechanism to ensure that a link is relatively stable before being brought into service. It can also be used effectively to limit the frequency at which link state transition events can occur. The default value for this leaf is determined by the underlying network device.

3.3. Dampening

The dampening feature introduces a configurable exponential decay mechanism to suppress the effects of excessive interface link state flapping. This feature allows the network operator to configure a device to automatically identify and selectively dampen a local interface which is flapping. Dampening an interface keeps the interface operationally down until the interface stops flapping and becomes stable. Configuring the dampening feature can improve convergence times and stability throughout the network by isolating failures so that disturbances are not propagated, which reduces the utilization of system processing resources by other devices in the network and improves overall network stability.

The basic algorithm uses a counter that is nominally increased by 1000 units every time the underlying interface link state changes from up to down. If the counter increases above the suppress threshold then the interface is kept down (and out of service) until either the maximum suppression time is reached, or the counter has reduced below the reuse threshold. The half-life period determines that rate at which the counter is periodically reduced. Implementations are not required to use a penalty of 1000 units in their dampening algorithm, but should ensure that the Suppress Threshold and Reuse Threshold values are scaled relative to the nominal 1000 unit penalty to ensure that the same configuration values provide consistent behaviour. The configurable values are described in more detail below.

3.3.1. Suppress Threshold

The suppress threshold is the value of the accumulated penalty that triggers the device to dampen a flapping interface. The flapping interface is identified by the device and assigned a penalty for each up to down link state change, but the interface is not automatically dampened. The device tracks the penalties that a flapping interface accumulates. When the accumulated penalty reaches the default or configured suppress threshold, the interface is placed in a dampened state.

3.3.2. Half-Life Period

The half-life period determines how fast the accumulated penalties can decay exponentially. Any penalties that have been accumulated on a flapping interface are reduced by half after each half-life period.

3.3.3. Reuse Threshold

If, after one or more half-life periods, the accumulated penalty decreases below the reuse threshold and the underlying interface link state is up then the interface is taken out of dampened state and allowed to go up.

3.3.4. Maximum Suppress Time

The maximum suppress time represents the maximum amount of time an interface can remain dampened when a penalty is assigned to an interface. The default of the maximum suppress timer is four times the half-life period. The maximum value of the accumulated penalty is calculated using the maximum suppress time, reuse threshold and half-life period.

3.4. Encapsulation

The encapsulation container holds a choice node that is to be augmented with datalink layer specific encapsulations, such as HDLC, PPP, or sub-interface 802.1Q tag match encapsulations. It ensures that an interface can only have a single datalink layer protocol configured.

3.5. Loopback

The loopback configuration leaf allows any physical interface to be configured to be in one of the possible following physical loopback modes, i.e. internal loopback, line loopback, or use of an external loopback connector. The use of YANG identities allows for the model to be extended with other modes of loopback if required.

3.6. MTU

Two MTU configuration leaves are provided to program the layer 2 interface in two different ways. Different mechanisms are provided to reflect the fact that devices handle their MTU configuration in different ways. A given device would only normally be expected to support MTU configuration using one of these mechanisms.

The preferable way to configure MTU is using the l2-mtu leaf that specifies the maximum size of a layer 2 frame including header and

payload, but excluding any frame check sequence (FCS) bytes. The payload MTU available to higher layer protocols is calculated from the l2-mtu after taking the layer 2 header size into account.

For Ethernet interfaces carrying 802.1Q VLAN tagged frames, the l2-mtu excludes the 4-8 byte overhead of any known (e.g. explicitly matched by a child sub-interface) 801.1Q VLAN tags.

The alternative way to configure MTU is using the l3-mtu leaf that specifies the maximum size of payload carried by a layer 2 frame. The maximum size of the layer 2 frame can then be derived by adding on the size of the layer 2 header overheads.

Note for reviewers: Is it correct/beneficial to support l3-mtu? It would be easier for clients if they only had a single MTU that they could configure. Can all devices sensibly handle an l2-mtu configuration leaf?

3.7. Sub-interface

The sub-interface feature specifies the minimal leaves required to define a child interface that is parented to another interface.

A sub-interface is a logical interface that handles a subset of the traffic on the parent interface. Separate configuration leaves are used to classify the subset of ingress traffic received on the parent interface to be processed in the context of a given sub-interface. All egress traffic processed on a sub-interface is given to the parent interface for transmission. Otherwise, a sub-interface is like any other interface in /if:interfaces and supports the standard interface features and configuration.

For some vendor specific interface naming conventions the name of the child interface is sufficient to determine the parent interface, which implies that the child interface can never be reparented to a different parent interface after it has been created without deleting the existing the sub-interface and recreating a new sub-interface. Even in this case it is useful to have a well defined leaf to cleanly identify the parent interface.

The model also allows for arbitrarily named sub-interfaces by having an explicit parent-interface leaf define the child -> parent relationship. In this naming scenario it is also possible for implementations to allow for logical interfaces to be reparented to new parent interfaces without needing the sub-interface to be destroyed and recreated.

3.8. Transport Layer

The transport layer leaf provides additional information as to which layer an interface is logically operating and forwarding traffic at. The implication of this leaf is that for traffic forwarded at a given layer that any headers for lower layers are stripped off before the packet is forwarded at the given layer. Conversely, on egress any lower layer headers must be added to the packet before it is transmitted out of the interface.

This leaf can also be used as a simple mechanism to determine whether particular types of configuration are valid. E.g. a layer 2 QoS policy could ensure that it is only applied to a interface running at transport layer 2.

4. Interfaces Ethernet-Like Module

The Interfaces Ethernet-Like Module is a small module that contains all configuration and operational data that is common across interface types that use Ethernet framing as their datalink layer encapsulation.

This module currently contains leaves for the configuration and reporting of the operational MAC address and the burnt-in MAC address (BIA) associated with any interface using Ethernet framing.

The "interfaces-ethernet-like" YANG module has the following structure:

```
module: ietf-interfaces-ethernet-like
augment /if:interfaces/if:interface:
   +--rw ethernet-like
      +--rw mac-address?   yang:mac-address
augment /if:interfaces-state/if:interface:
   +--ro ethernet-like
      +--ro mac-address?       yang:mac-address
      +--ro bia-mac-address?   yang:mac-address
```

5. Interfaces Common YANG Module

This YANG module augments the interface container defined in RFC 7223 [RFC7223].

```
<CODE BEGINS> file "ietf-interfaces-common@2016-07-07.yang"
module ietf-interfaces-common {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-interfaces-common";
  prefix if-cmn;
```

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

     import iana-if-type {
       prefix ianaift;
     }

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

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

        WG Chair: Lou Berger
                  <mailto:lberger@labn.net>

        WG Chair: Kent Watsen
                  <mailto:kwatsen@juniper.net>

        Editor:   Robert Wilton
                  <mailto:rwilton@cisco.com>";

     description
       "This module contains common definitions for extending the IETF
        interface YANG model (RFC 7223) with common configurable layer 2
        properties.

        Copyright (c) 2016 IETF Trust and the persons identified as
        authors of the code.  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 XXX; see the RFC
        itself for full legal notices.";

     revision 2016-07-07 {
       description
         "Update module title and description text to IETF standard
          text";

       reference "Internet draft: draft-ietf-netmod-intf-ext-yang-01";
```

```
     }

     revision 2015-10-19 {
       description
         "Add support for various common interface configuration
          parameters that are likely to be widely implemented by various
          network device vendors.";

       reference "Internet draft: draft-wilton-netmod-intf-ext-yang-01";
     }

     feature bandwidth {
       description
         "This feature indicates that the device supports configurable
          interface bandwidth.";
       reference "Section 3.1 Bandwidth";
     }

     feature carrier-delay {
       description
         "This feature indicates that configurable interface
          carrier delay is supported, which is a feature is used to
          limit the propagation of very short interface link state
          flaps.";
       reference "Section 3.2 Carrier Delay";
     }

     feature dampening {
       description
         "This feature indicates that the device supports interface
          dampening, which is a feature that is used to limit the
          propagation of interface link state flaps over longer
          periods";
       reference "Section 3.3 Dampening";
     }

     feature loopback {
       description
         "This feature indicates that configurable interface loopback
          is supported.";
       reference "Section 3.5 Loopback";
     }

     feature configurable-l2-mtu {
       description
         "This feature indicates that the device supports configuring
          layer 2 MTUs on interfaces.  Such MTU configurations include
          the layer 2 header overheads (but exclude any FCS overhead).
```

```
          The payload MTU available to higher layer protocols is either
          derived from the layer 2 MTU, taking into account the size of
          the layer 2 header, or is further restricted by explicit layer
          3 or protocol specific MTU configuration.";
       reference "Section 3.6 MTU";
     }

     feature sub-interfaces {
       description
         "This feature indicates that the device supports the
          instantiation of sub-interfaces.  Sub-interfaces are defined
          as logical child interfaces that allow features and forwarding
          decisions to be applied to a subset of the traffic processed
          on the specified parent interface.";
       reference "Section 3.7 Sub-interface";
     }

     feature transport-layer {
       description
         "This feature indicates that the device supports configurable
          transport layer.";
       reference "Section 3.8 Transport Layer";
     }

     /*
      * Define common identities to help allow interface types to be
      * assigned properties.
      */
     identity sub-interface {
       description "Base type for generic sub-interfaces.  New or custom
                    interface types can derive from this type to
                    inherit generic sub-interface configuration";
     }

     identity ethSubInterface{
       base ianaift:l2vlan;
       base sub-interface;

       description "Sub-interface of any interface types that uses
                    Ethernet framing (with or without 802.1Q tagging)";
     }

     identity loopback {
       description "Base type for interface loopback options";
     }
     identity loopback-internal {
       base loopback;
       description "All egress traffic on the interface is internally
```

```
                   looped back within the interface to be received on
                   the ingress path.";
     }
     identity loopback-line {
       base loopback;
       description "All ingress traffic received on the interface is
                   internally looped back within the interface to the
                   egress path.";
     }
     identity loopback-connector {
       base loopback;
       description "The interface has a physical loopback connector
                   attached to that loops all egress traffic back into
                   the interface's ingress path, with equivalent
                   semantics to loopback-internal";
     }

     /*
      * Augments the IETF interfaces model with a leaf to explicitly
      * specify the bandwidth available on an interface.
      */
     augment "/if:interfaces/if:interface" {
       if-feature "bandwidth";

       description "Add a top level node for interface bandwidth.";
       leaf bandwidth {
         type uint64;
         units kbps;
         description
           "The bandwidth available on the interface in Kb/s. This
            configuration is used by routing protocols to adjust the
            metrics associated with the interface, but does not limit
            the amount of traffic that can be sent or received on the
            interface. A separate QoS policy would need to be configured
            to limit the ingress or egress traffic. If not configured,
            the default bandwidth is the maximum available bandwidth of
            the underlying interface.";
       }
     }

     /*
      * Defines standard YANG for the Carrier Delay feature.
      */
     augment "/if:interfaces/if:interface" {
       if-feature "carrier-delay";
       description "Augments the IETF interface model with
                   carrier delay configuration for interfaces that
                   support it.";
```

```
       container carrier-delay {
         description "Holds carrier delay related feature
                      configuration";
         leaf down {
           type uint32;
           units milliseconds;
           description
             "Delays the propagation of a 'loss of carrier signal' event
              that would cause the interface state to go down, i.e. the
              command allows short link flaps to be suppressed. The
              configured value indicates the minimum time interval (in
              milliseconds) that the carrier signal must be continuously
              down before the interface state is brought down. If not
              configured, the behaviour on loss of carrier signal is
              vendor/interface specific, but with the general
              expectation that there should be little or no delay.";
         }
         leaf up {
           type uint32;
             units milliseconds;
             description
               "Defines the minimum time interval (in milliseconds) that
                the carrier signal must be continuously present and
                error free before the interface state is allowed to
                transition from down to up.  If not configured, the
                behaviour is vendor/interface specific, but with the
                general expectation that sufficient default delay
                should be used to ensure that the interface is stable
                when enabled before being reported as being up.
                Configured values that are too low for the hardware
                capabilties may be rejected.";
         }
       }
     }

     /*
      * Augments the IETF interfaces model with a container to hold
      * generic interface dampening
      */
     augment "/if:interfaces/if:interface" {
       if-feature "dampening";
       description
         "Add a container for interface dampening configuration";

       container dampening {
         presence "Enable interface link flap dampening with default
                   settings (that are vendor/device specific)";
         description "Interface dampening limits the propagation of
```

```
                      interface link state flaps over longer periods";
          leaf half-life {
            type uint32;
            units seconds;
            description
              "The Time (in seconds) after which a penalty reaches half
               its original value. Once the interface has been assigned
               a penalty, the penalty is decreased by half after the
               half-life period. For some devices, the allowed values may
               be restricted to particular multiples of seconds. The
               default value is vendor/device specific.";
          }

          leaf reuse {
            type uint32;
            description
              "Penalty value below which a stable interface is
               unsuppressed (i.e. brought up) (no units).  The default
               value is vendor/device specific.  The penalty value for a
               link up->down state change is nominally 1000 units.";
          }

          leaf suppress {
            type uint32;
            description
              "Limit at which an interface is suppressed (i.e. held down)
               when its penalty exceeds that limit (no units). The value
               must be greater than the reuse threshold.  The default
               value is vendor/device specific.  The penalty value for a
               link up->down state change is nominally 1000 units.";
          }

          leaf max-suppress-time {
            type uint32;
            units seconds;
            description
              "Maximum time (in seconds) that an interface can be
               suppressed. This value effectively acts as a ceiling that
               the penalty value cannot exceed.  The default value is
               vendor/device specific.";
          }
        }
      }

      /*
       * Various types of interfaces support a configurable layer 2
       * encapsulation, any that are supported by YANG should be
       * listed here.
```

```
     *
     * Different encapsulations can hook into the common encaps-type
     * choice statement.
     */
    augment "/if:interfaces/if:interface" {
      when "if:type = 'ianaift:ethernetCsmacd' or
            if:type = 'ianaift:ieee8023adLag' or
            if:type = 'ethSubInterface' or
            if:type = 'ianaift:pos' or
            if:type = 'ianaift:atmSubInterface'"  {
        description "All interface types that can have a configurable
                     L2 encapsulation";
        /*
         * TODO - Should we introduce an abstract type to make this
         *        extensible to new interface types, or vendor specific
         *        interface types?
         */
      }

      description "Add encapsulation top level node to interface types
                   that support a configurable L2 encapsulation";

      container encapsulation {
        description
          "Holds the L2 encapsulation associated with an interface";
        choice encaps-type {
          description "Extensible choice of L2 encapsulations";
        }
      }
    }

    /*
     * Various types of interfaces support loopback configuration, any
     * that are supported by YANG should be listed here.
     */
    augment "/if:interfaces/if:interface" {
      when "if:type = 'ianaift:ethernetCsmacd' or
            if:type = 'ianaift:sonet' or
            if:type = 'ianaift:atm' or
            if:type = 'ianaift:otnOtu'" {
        description
          "All interface types that support loopback configuration.";
      }
      if-feature "loopback";
      description "Augments the IETF interface model with loopback
                   configuration for interfaces that support it.";

      leaf loopback {
```

```
         type identityref {
           base loopback;
         }
         description "Enables traffic loopback.";
       }
     }

     /*
      * Many types of interfaces support a configurable layer 2 MTU.
      */
     augment "/if:interfaces/if:interface" {
       description "Add configurable layer 2 MTU to all appropriate
                    interface types.";

       leaf l2-mtu {
         if-feature "configurable-l2-mtu";
         type uint16 {
           range "64 .. 65535";
         }
         description
           "The maximum size of layer 2 frames that may be transmitted
            or received on the interface (excluding any FCS overhead).
            In the case of Ethernet interfaces it also excludes the
            4-8 byte overhead of any known (i.e. explicitly matched by
            a child sub-interface) 801.1Q VLAN tags.";
       }
     }

     /*
      * Add generic support for sub-interfaces.
      *
      * This should be extended to cover all interface types that are
      * child interfaces of other interfaces.
      */
     augment "/if:interfaces/if:interface" {
       when "derived-from(if:type, 'sub-interface') or
             if:type = 'ianaift:atmSubInterface' or
             if:type = 'ianaift:frameRelay'"  {
         description
           "Any ianaift:types that explicitly represent sub-interfaces
            or any types that derive from the sub-interface identity";
       }
       if-feature "sub-interfaces";
       description "Add a parent interface field to interfaces that
                    model sub-interfaces";
       leaf parent-interface {
         type if:interface-ref;
```

```
         mandatory true;
         description
           "This is the reference to the parent interface of this
            sub-interface.";
       }
     }

     /*
      * Augments the IETF interfaces model with a leaf that indicates
      * which layer traffic is to be transported at.
      */
     augment "/if:interfaces/if:interface" {
       if-feature "transport-layer";
       description "Add a top level node to appropriate interfaces to
                    indicate which tranport layer an interface is
                    operating at";

       leaf transport-layer {
         type enumeration {
           enum layer-1 {
             value 1;
             description "Layer 1 transport.";
           }
           enum layer-2 {
             value 2;
             description "Layer 2 transport";
           }
           enum layer-3 {
             value 3;
             description "Layer 3 transport";
           }
         }
         default layer-3;
         description
           "The transport layer at which the interface is operating at";
       }
     }
   }
   <CODE ENDS>
```

6. Interfaces Ethernet-Like YANG Module

This YANG module augments the interface container defined in RFC 7223 [RFC7223] for Etherlike interfaces. This includes Ethernet interfaces, 802.3 LAG (802.1AX) interfaces, VLAN sub-interfaces, Switch Virtual interfaces, and Pseudo-Wire Head-End interfaces.

```
<CODE BEGINS> file "ietf-interfaces-ethernet-like@2016-07-07.yang"
```

```
module ietf-interfaces-ethernet-like {
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-interfaces-ethernet-like";

  prefix ethlike;

  import ietf-interfaces {
    prefix if;
  }

  import ietf-yang-types {
    prefix yang;
  }

  import iana-if-type {
    prefix ianaift;
  }

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

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

     WG Chair: Lou Berger
               <mailto:lberger@labn.net>

     WG Chair: Kent Watsen
               <mailto:kwatsen@juniper.net>

     Editor:   Robert Wilton
               <mailto:rwilton@cisco.com>";

  description
    "This module contains YANG definitions for configuration for
     'Ethernet-like' interfaces.  It is applicable to all interface
     types that use Ethernet framing and expose an Ethernet MAC
     layer, and includes such interfaces as physical Ethernet
     interfaces, Ethernet LAG interfaces and VLAN sub-interfaces.

     Copyright (c) 2016 IETF Trust and the persons identified as
     authors of the code.  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 XXX; see the RFC
        itself for full legal notices.";

     revision 2016-07-07 {
       description
         "Update module title and description text to IETF standard
          text";

       reference "Internet draft: draft-ietf-netmod-intf-ext-yang-01";
     }

     revision 2015-06-26 {
       description "Updated reference to new internet draft name.";

       reference
         "Internet draft: draft-wilton-netmod-intf-ext-yang-00";
     }

     /*
      * Configuration parameters for Etherlike interfaces.
      */
     augment "/if:interfaces/if:interface" {
       when "if:type = 'ianaift:ethernetCsmacd' or
             if:type = 'ianaift:ieee8023adLag' or
             if:type = 'ianaift:l2vlan' or
             if:type = 'ianaift:ifPwType'"   {
         description "Applies to all Ethernet-like interfaces";
       }
       description
         "Augment the interface model with configuration parameters for
          all Ethernet-like interfaces";

       container ethernet-like {
         description "Contains configuration parameters for interfaces
                      that use Ethernet framing and expose an Ethernet
                      MAC layer.";
         leaf mac-address {
           type yang:mac-address;
           description
             "The configured MAC address of the interface.";
         }
       }
     }

     /*
```

```
      * Operational state for Etherlike interfaces.
      */
     augment "/if:interfaces-state/if:interface" {
       when "if:type = 'ianaift:ethernetCsmacd' or
             if:type = 'ianaift:ieee8023adLag' or
             if:type = 'ianaift:l2vlan' or
             if:type = 'ianaift:ifPwType'"   {
         description "Applies to all Ethernet-like interfaces";
       }
       description
         "Augments the interface model with operational state parameters
          for all Ethernet-like interfaces.";
       container ethernet-like {
         description "Contains operational state parameters for
                      interfaces that use Ethernet framing and expose an
                      Ethernet MAC layer.";
         leaf mac-address {
           type yang:mac-address;
           description
             "The operational MAC address of the interface, if
              applicable";
         }

         leaf bia-mac-address {
           type yang:mac-address;
           description
             "The 'burnt-in' MAC address.  I.e the default MAC address
              assigned to the interface if none is explicitly
              configured.";
         }
       }
     }
   }
   <CODE ENDS>
```

7. Acknowledgements

The authors wish to thank Juergen Schoenwaelder, Mahesh Jethanandani, Michael Zitao, Neil Ketley and Qin Wu for their helpful comments contributing to this draft.

8. IANA Considerations

This document defines several new YANG module and the authors politely request that IANA assigns unique names to the YANG module files contained within this draft, and also appropriate URIs in the "IETF XML Registry".

9. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol RFC 6241 [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory to implement secure transport is SSH RFC 6242 [RFC6242]. The NETCONF access control model RFC 6536 [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 this 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:

9.1. interfaces-common.yang

The interfaces-common YANG module contains various configuration leaves that affect the behavior of interfaces. Modifying these leaves can cause an interface to go down, or become unreliable, or to drop traffic forwarded over it. More specific details of the possible failure modes are given below.

The following leaf could cause the interface to go down, and stop processing any ingress or egress traffic on the interface:

o /if:interfaces/if:interface/loopback

The following leaf could cause changes to the routing metrics. Any change in routing metrics could cause too much traffic to be routed through the interface, or through other interfaces in the network, potentially causing traffic loss due to excesssive traffic on a particular interface or network device:

o /if:interfaces/if:interface/bandwidth

The following leaves could cause instabilities at the interface link layer, and cause unwanted higher layer routing path changes if the leaves are modified, although they would generally only affect a device that had some underlying link stability issues:

o /if:interfaces/if:interface/carrier-delay/down

o /if:interfaces/if:interface/carrier-delay/up

- o /if:interfaces/if:interface/dampening/half-life

- o /if:interfaces/if:interface/dampening/reuse

- o /if:interfaces/if:interface/dampening/suppress

- o /if:interfaces/if:interface/dampening/max-suppress-time

The following leaves could cause traffic loss on the interface because the received or transmitted frames do not comply with the frame matching criteria on the interface and hence would be dropped:

- o /if:interfaces/if:interface/encapsulation

- o /if:interfaces/if:interface/l2-mtu

- o /if:interfaces/if:interface/l3-mtu

- o /if:interfaces/if:interface/transport-layer

Normally devices will not allow the parent-interface leaf to be changed after the interfce has been created. If an implementation did allow the parent-interface leaf to be changed then it could cause all traffic on the affected interface to be dropped. The affected leaf is:

- o /if:interfaces/if:interface/parent-interface

9.2. interfaces-ethernet-like.yang

Generally, the configuration nodes in the interfaces-ethernet-like YANG module are concerned with configuration that is common across all types of Ethernet-like interfaces. Currently, the module only contains a node for configuring the operational MAC address to use on an interface. Adding/modifying/deleting this leaf has the potential risk of causing protocol instability, excessive protocol traffic, and general traffic loss, particularly if the configuration change caused a duplicate MAC address to be present on the local network . The following leaf is affected:

- o interfaces/interface/ethernet-like/mac-address

10. References

10.1. Normative References

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

[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>.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <http://www.rfc-editor.org/info/rfc7223>.

[RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <http://www.rfc-editor.org/info/rfc7224>.

10.2. Informative References

[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>.

Authors' Addresses

Robert Wilton (editor)
Cisco Systems

Email: rwilton@cisco.com

David Ball
Cisco Systems

Email: daviball@cisco.com

Tapraj Singh
Cisco Systems

Email: tapsingh@juniper.net

Selvakumar Sivaraj
Juniper Networks

Email: ssivaraj@juniper.net