YANG models for 802.3

Yan Zhuang, Huawei Technologoies
Marek Hajduczenia, Bright House Networks

IEEE 802.3 Interim meeting

Atlanta, US

January 17th – 22nd, 2016

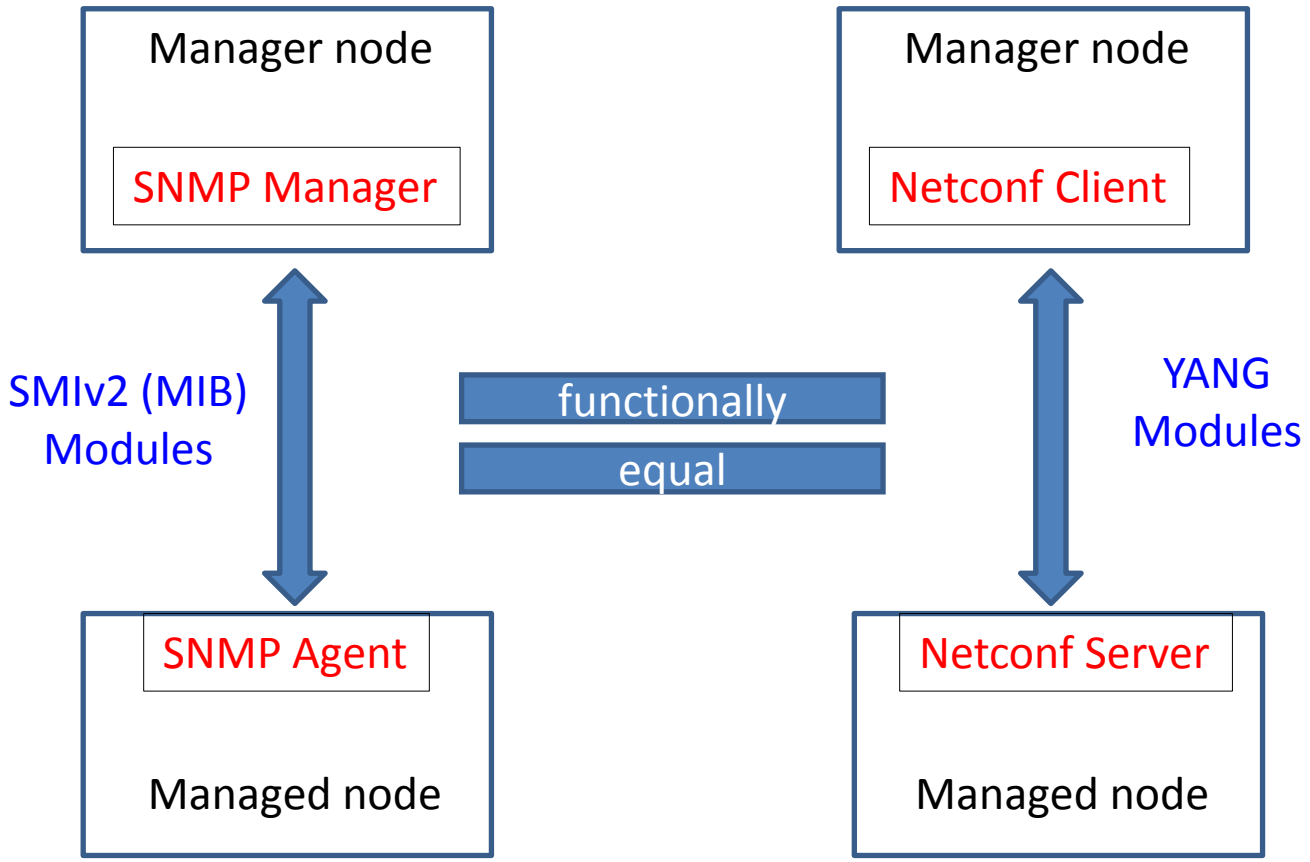
Today's Topics

- YANG Introduction
- Marketing Trends: the move to YANG
- YANG for IEEE 802.3
- Why now?
- Discussions

Section I:

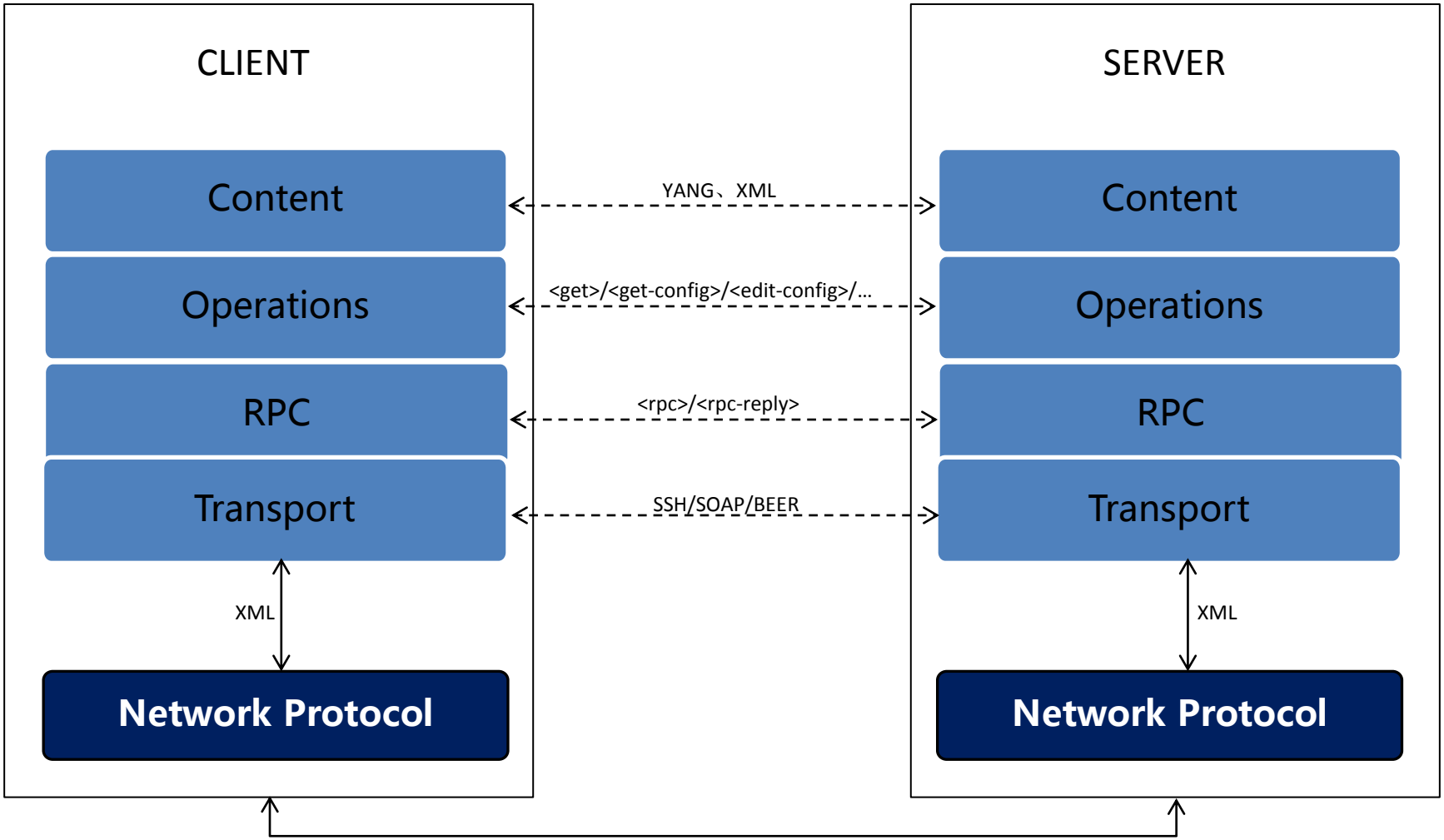
YANG INTRODUCTION

Network Management: SNMP+MIB vs Netconf+YANG



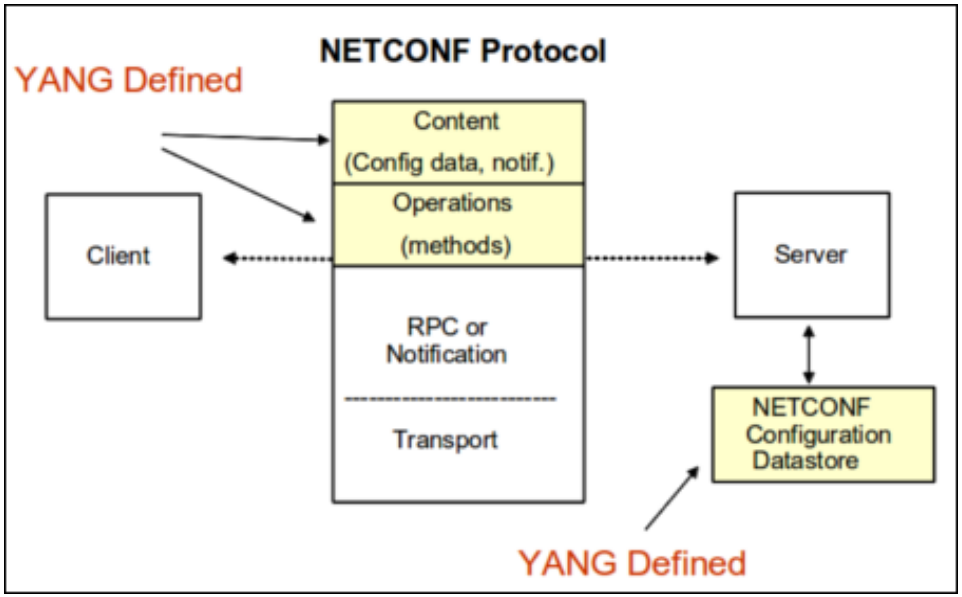
- Similar management framework and protocol message models
- Different module structure and schema
- YANG models can also be transmitted in RESTCONF, but we take NETCONF as example to explain how to use YANG.

NETCONF



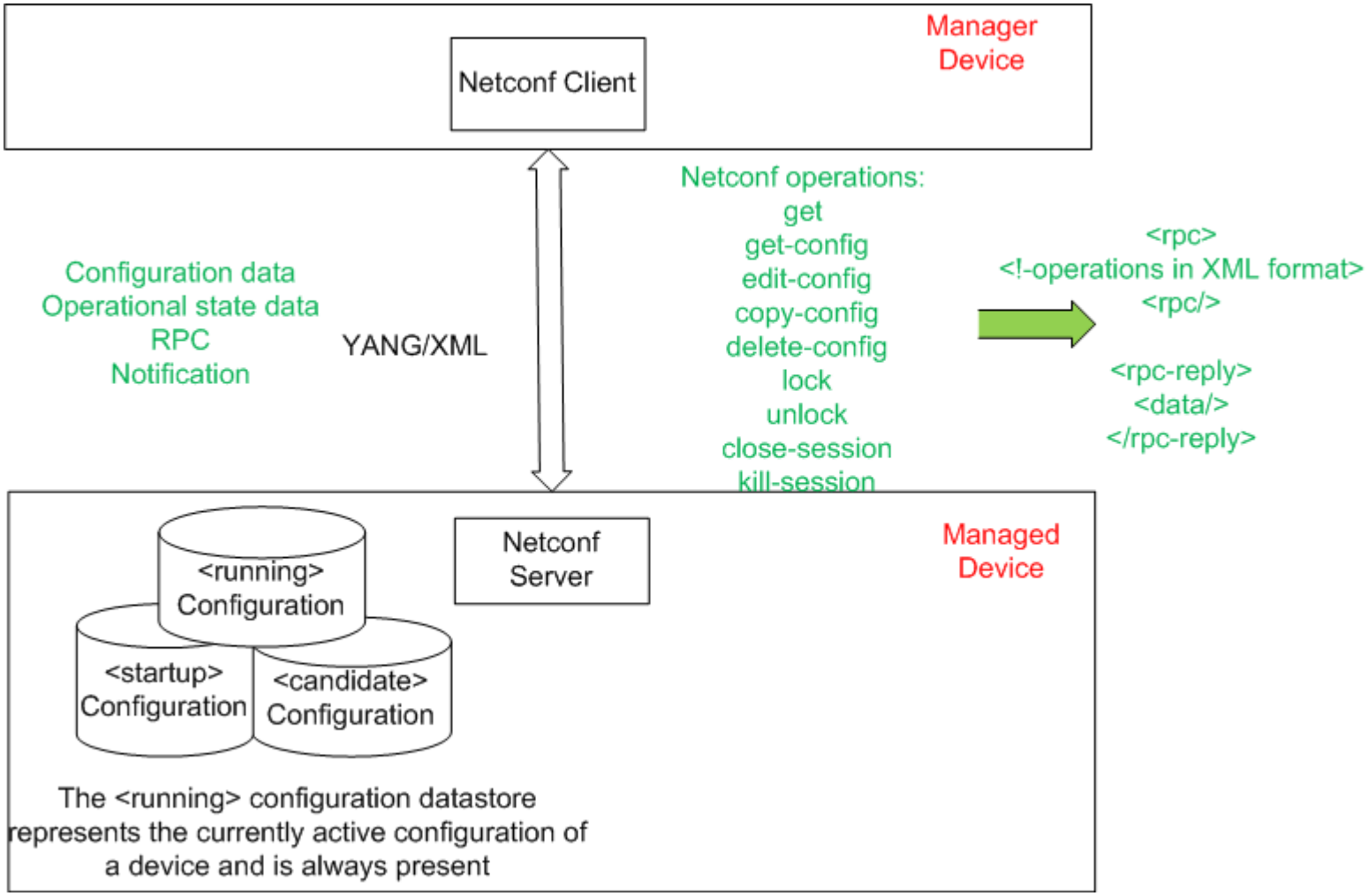
- Transaction-based management protocol
- Uses SSH/SOAP/BEER, etc., for data transfer between client and server

YANG + NETCONF



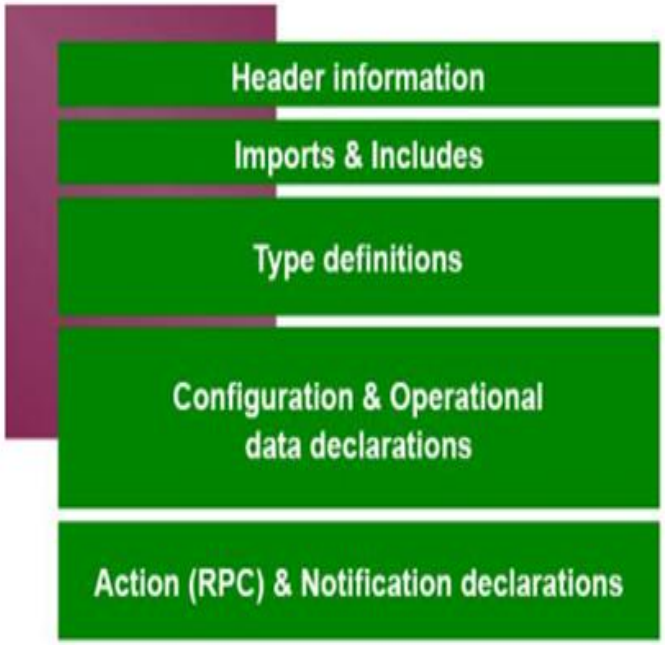
- The YANG model describes …
  - configuration actions
  - monitoring capabilities
  - admin actions
  - notifications and events
- … for each device type and version

Deployment Model



YANG Module – Basic Elements

- YANG is a ***Data Modeling Language***
- The module body includes several basic components:



- YANG model will be directly mapped to XML content and transmitted in protocols such as NETCONF/RESTCONF.

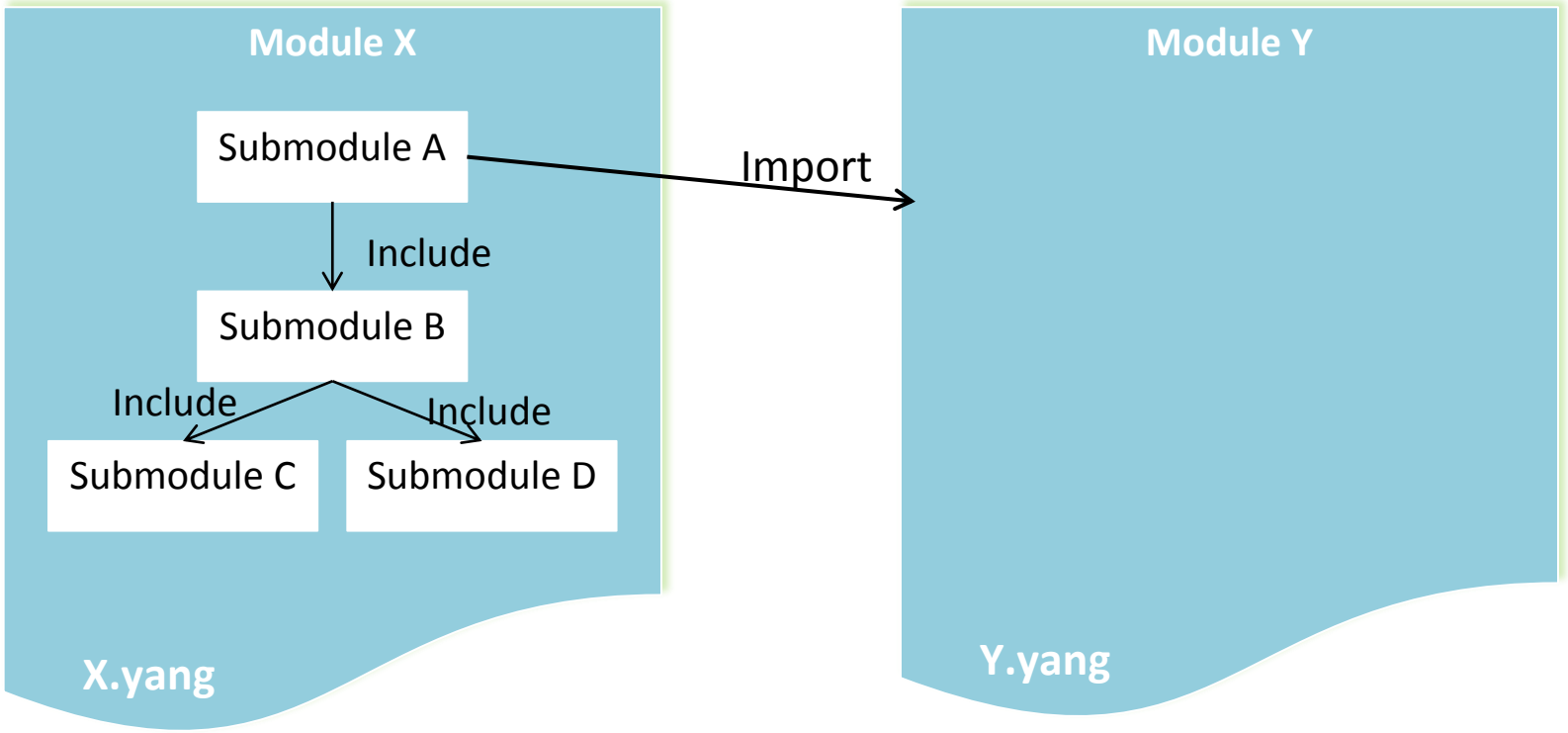
1 - Header information

- Code example for Ietf-interfaces module

```
1  module ietf-interfaces {
2
3       namespace "urn:ietf:params:xml:ns:yang:ietf-interfaces";
4       prefix if;
5
6       import ietf-yang-types {
7         prefix yang;
8       }
9
10      organization
11        "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
12
13      contact
14        "WG Web:   <http://tools.ietf.org/wg/netmod/>...";
15
16        //module description
17      description
18        "This module contains a collection of YANG definitions for
19         managing network interfaces.
20
21         Copyright (c) 2014 IETF Trust and the persons identified as
22         authors of the code.  All rights reserved";
23
24
25         //revision info.
26         revision 2014-05-08 {
27       }
```

2 - Imports and Includes

- Available to reuse type names, groupings, other objects, etc., defined in external YANG modules.



Include: from a different submodule

Import: from a different module

That is:
If A wants to use types defined in module Y, it should "import Y {prefix Y}" and use "Y:typeY;"
If A wants to use types in submodule B, it should " include B" and use "type typeB or X:typeB;

3 – Type Definition

- The "type" statement is used to put further restrictions on the YANG built-in or a derived type.
- The "typedef" statement defines a new data type
  - may be used locally in the module or submodule
  - may be used by other modules that import it

```
typedef percent {
        type uint8 {
        range "0 .. 100";
        }
        description "Percentage";
}
leaf completed {
        type percent;
}
```

ietf-interfaces type definitions:

```
56      typedef interface-ref {
57        type leafref {
58          path "/if:interfaces/if:interface/if:name";
59        }
60        description
61          "This type is used by data models that need to reference
62           configured interfaces.";
63      }
64
65      typedef interface-state-ref {
66        type leafref {
67          path "/if:interfaces-state/if:interface/if:name";
68        }
69        description
70          "This type is used by data models that need to reference
71           the operationally present interfaces.";
72      }
```

3 – Built-in Types

Name	Description
binary	Any binary data
bits	A set of bits or flags
boolean	"true" or "false"
decimal64	64-bit signed decimal number
empty	A leaf that does not have any value
enumeration	Enumerated strings
identityref	A reference to an abstract identity
instance-identifier	References a data tree node
int8	8-bit signed integer
int16	16-bit signed integer
int32	32-bit signed integer
int64	64-bit signed integer
leafref	A reference to a leaf instance
string	Human-readable string
uint8	8-bit unsigned integer
uint16	16-bit unsigned integer
uint32	32-bit unsigned integer
uint64	64-bit unsigned integer
union	Choice of member types

4 - Configuration and Operational data

- The configuration and operational data are composed of data/schema nodes to carry configuration and state parameters

Ietf-interfaces configuration and operational data declaration:

```
62      /*
63       * Configuration data nodes
64       */
65
66      container interfaces {
67         description
68          "Interface configuration parameters.";
69        //...
70      }
71
72      /*
73       * Operational state data nodes
74       */
75
76      container interfaces-state {
77       config false;
78        description
79          "Data nodes for the operational state of interfaces.";
80        //....
81      }
82    }
```

Configuration data

State data

5 – Data Nodes (Leaf/Leaf-List)

- Leaf Nodes
  - A leaf instance contains simple data like an integer or a string.
  - It has exactly one value of a particular type and no child nodes

```
83      leaf description {
84            type string;
85            description
86              "A textual description of the interface.";
87            reference
88              "RFC 2863: The Interfaces Group MIB - ifAlias";
89          }
```

- Leaf-List nodes
  - A leaf-list defines a sequence of values of a particular type.

```
443         leaf-list higher-layer-if {
444           type interface-state-ref;
445           description
446             "A list of references to interfaces layered on top of this
447              interface.";
448           reference
449             "RFC 2863: The Interfaces Group MIB - ifStackTable";
450         }
```

5 – Data Nodes (Container / List)

- Container Nodes
  - groups related nodes in a subtree.
  - A container has only child nodes and no value.
  - A container contains any number of child nodes of any type.
- List Nodes
  - Defines a sequence of list entries.
  - Each entry is like a structure / record instance, and uniquely identified by values of key leaves.
  - A list can define multiple key leaves and contain any number of child nodes of any type.

```
container interfaces {
   description
     "Interface configuration parameters.";

   list interface {
     key "name";

     description
       "The list of configured interfaces on the device.";

    leaf name {
      type string;
      description
        "The name of the interface.";
     }

     leaf description {
       type string;
       description
         "A textual description of the interface.";
     }

     leaf type {
       type identityref {
         base interface-type;
       }
       mandatory true;
       description
         "The type of the interface.";
     }

     leaf enabled {
       type boolean;
       default "true";
       description
         "This leaf contains the configured, desired state of the
          interface.";
     }
     //...
   }
}
```

The interfaces container contains a list of interface, each of which includes name, description, type etc al.

6 – Choice Nodes

- The "choice" statement contains a set of "case" statements that define sets of schema nodes that cannot appear together.
- Each "case" may contain multiple nodes, but each node may appear in only one "case" under a "choice".

```
container food {
  choice snack {
    case sports-arena {
      leaf pretzel {
        type empty;
      }
      leaf beer {
        type empty;
      }
    }
    case late-night {
      leaf chocolate {
        type enumeration {
          enum dark;
          enum milk;
          enum first-available;
        }
      }
    }
  }
}
```

7 – Operations (RPC / Action)

- RPC:
  - The operations' name, input parameters, and output parameters are modeled using YANG data definition statements.
  - Operations on the top-level in a module are defined with the "rpc" statement.

```
rpc activate-software-image {
  input {
    leaf image-name {
      type string;
    }
  }
  output {
    leaf status {
      type string;
    }
  }
}
```

- Action:
  - Operations can also be tied to a data node.
  - Such operations are defined with the "action" statement.

```
list interface {
  key "name";

  leaf name {
    type string;
  }

  action ping {
    input {
      leaf destination {
        type inet:ip-address;
      }
    }
    output {
      leaf packet-loss {
        type uint8;
      }
    }
  }
}
```

8 – Notifications

- Used to model the content of a notification event associated with the specific event

```
notification link-failure {
  description
    "A link failure has been detected";
  leaf if-name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf if-admin-status {
    type admin-status;
  }
  leaf if-oper-status {
    type oper-status;
  }
}
```

XML Encoding:

```
<notification>
   <link-failure>
     <if-name>eth0</if-name>
     <if-admin-status>up</if-admin-status>
     <if-oper-status>up</if-oper-status>
    </link-failure>
</notification>
```

9 – Extending existing modules

- To Insert additional nodes into data models, including both the current module (and its submodules) or an external module.
- Using “augment” statement to generate a new model based on existing models with additional data nodes.

Code to augment ietf-interfaces with IEEE 802.3 parameters

```
 augment "/if:interfaces/if:interface" {
  when "if:type = 'ianaift:ethernetCsmacd'" {
    description "Applies to all P2P Ethernet interfaces";
  }
  description
"Augment interface model with IEEE Std 803.2 Ethernet
 specific configuration nodes";

  container ethernet {
          //ethernet interface related configurations
    }
  }
```

“augment”: Where to insert new data nodes

“when”: the condition that the new defined data nodes are valid

Defined a new container to hold parameters

The data hierarchy of this augmentation:

```
module: ethernet
augment /if:interfaces/if:interface:
   +--rw ethernet
```

(example) of data hierarchy to ietf-interfaces module

configuration

Operational state

```
module: ietf-interfaces
  +--rw interfaces
  |  +--rw interface* [name]
  |     +--rw name                        string
  |     +--rw description?                string
  |     +--rw type                        identityref
  |     +--rw enabled?                    boolean
  |     +--rw link-up-down-trap-enable?   enumeration {if-mib}?
  +--ro interfaces-state
     +--ro interface* [name]
        +--ro name                string
        +--ro type                identityref
        +--ro admin-status        enumeration {if-mib}?
        +--ro oper-status         enumeration
        +--ro last-change?        yang:date-and-time
        +--ro if-index            int32 {if-mib}?
        +--ro phys-address?       yang:phys-address
        +--ro higher-layer-if*    interface-state-ref
        +--ro lower-layer-if*     interface-state-ref
        +--ro speed?              yang:gauge64
        +--ro statistics
           +--ro discontinuity-time    yang:date-and-time
           +--ro in-octets?            yang:counter64
           +--ro in-unicast-pkts?      yang:counter64
           +--ro in-broadcast-pkts?    yang:counter64
           +--ro in-multicast-pkts?    yang:counter64
           +--ro in-discards?          yang:counter32
           +--ro in-errors?            yang:counter32
           +--ro in-unknown-protos?    yang:counter32
           +--ro out-octets?           yang:counter64
           +--ro out-unicast-pkts?     yang:counter64
           +--ro out-broadcast-pkts?   yang:counter64
           +--ro out-multicast-pkts?   yang:counter64
           +--ro out-discards?         yang:counter32
           +--ro out-errors?           yang:counter32
```

(example) of XML Encoding

- The following gives a corresponding XML instance for devices to implement the ietf-interfaces data models:

```
<if:interfaces: xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces" >
      <if:interface>
       <if:name>eth0</if:name>
       <if:type>ianaift:ethernetCsmacd</if:type>
       <if:description>
          Link to A.
       </if:description>
      </if:interface>
  </if:interfaces>
<interfaces-state xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces>
      <if: interface>
        <name>eth0</name>
         <type>ianaift:ethernetCsmacd</type>
           <admin-status>down</admin-status>
         <oper-status>down</oper-status>
           <if-index>2</if-index>
           <phys-address>00:01:02:03:04:05</phys-address>
           <statistics>
              <discontinuity-time> 2015-04-01T03:00:00+00:00 </discontinuity-time>
              <!-- counters now shown here -->
          </statistics>
       </interface>
  </interfaces-state>
```

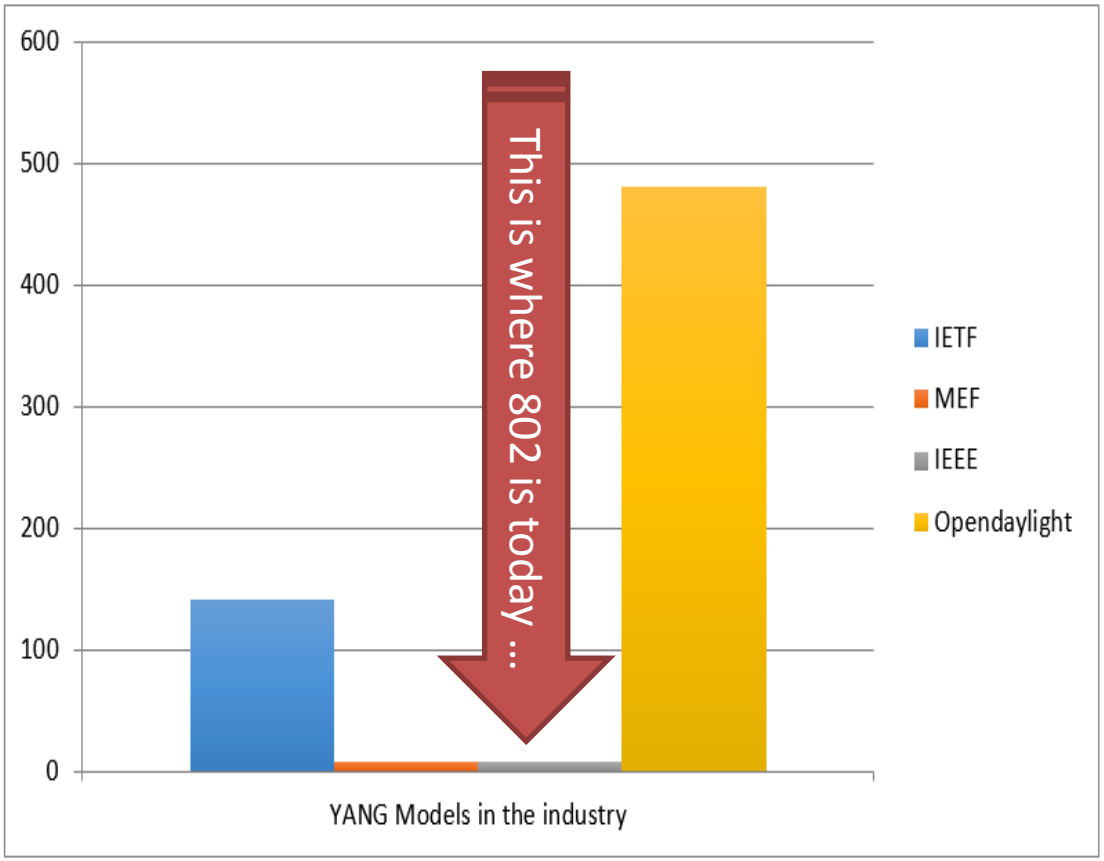
Section II

THE MOVE TO YANG

Why do we need YANG then?

- YANG+NETCONF/RESTCONF can provide …
  - More flexible and extensible device modeling (YANG)
  - Human-readable language syntax (YANG)
  - Configuration focused on end-to-end service, and not individual devices (NETCONF/RESTCOF)
  - Transaction-oriented exchange, with device state tracking, running and backup configurations, commit and rollback functions (NETCONF/YANG)
  - Configuration validation prior to committing changes (NETCONF/RESTCONF)
  - Support for multiple configurations per device for simpler rollback (YANG/NETCONF)
  - Common configuration and state model across all multi-vendor devices in the network(YANG).

SDOs are developing YANG models



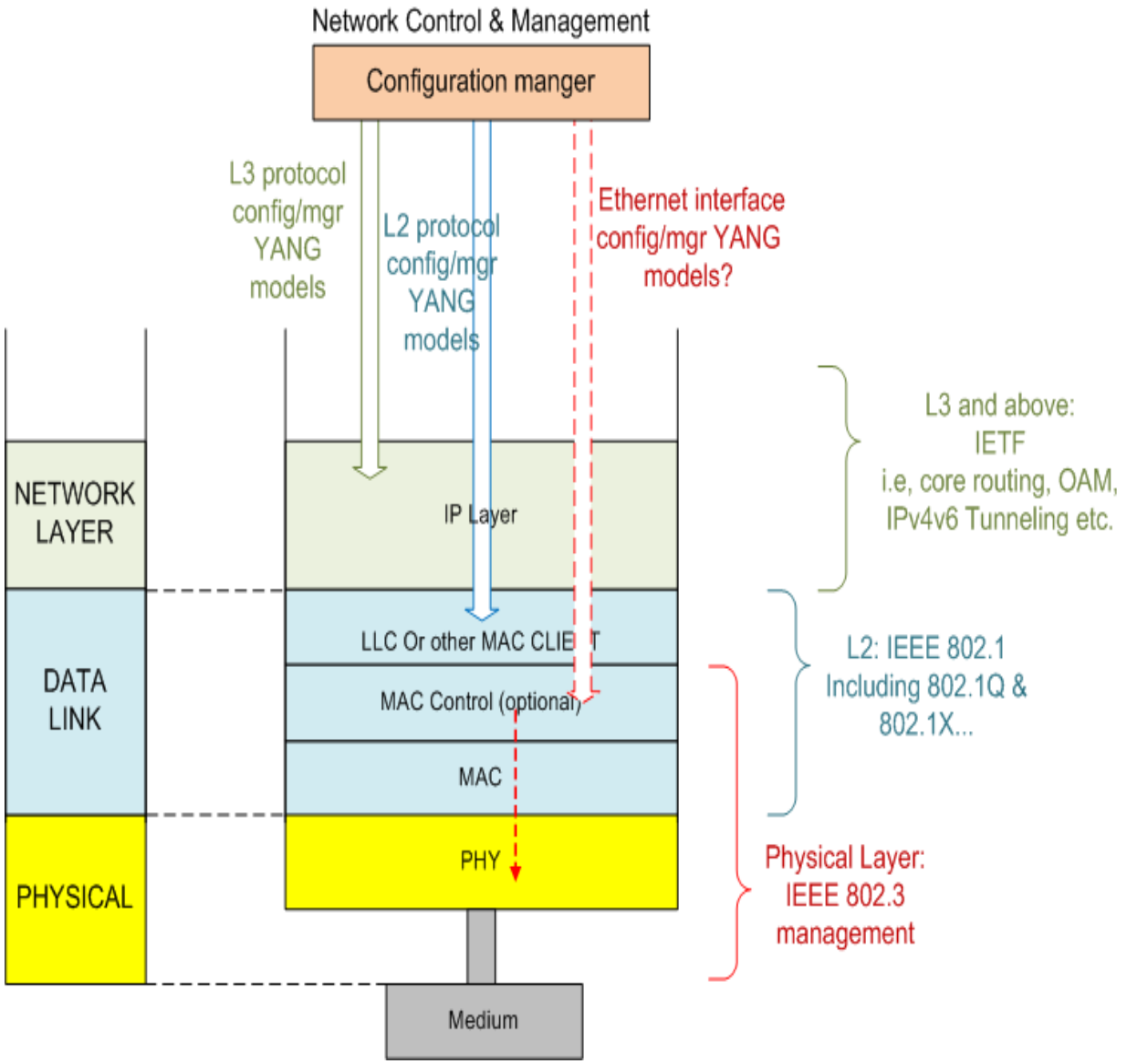
The development of YANG models are increasing in the industry and SDOs, including:

- IETF
- IEEE 802.1
- MEF
- ITU-T
- ODL
- BBF
- ….

From **YANG and NETCONF/RESTCONF Gain Traction in the Industry: Latest Status**, in IETF Journal, for IETF 94.

Given the pervasive character of 802.3 PHYs in different application areas, the availability of standardized YANG models is critical for the future.
If no standardized models are defined, private Ethernet-like YANG models will created by individual vendors, leading to interoperability problems.

Different YANG models

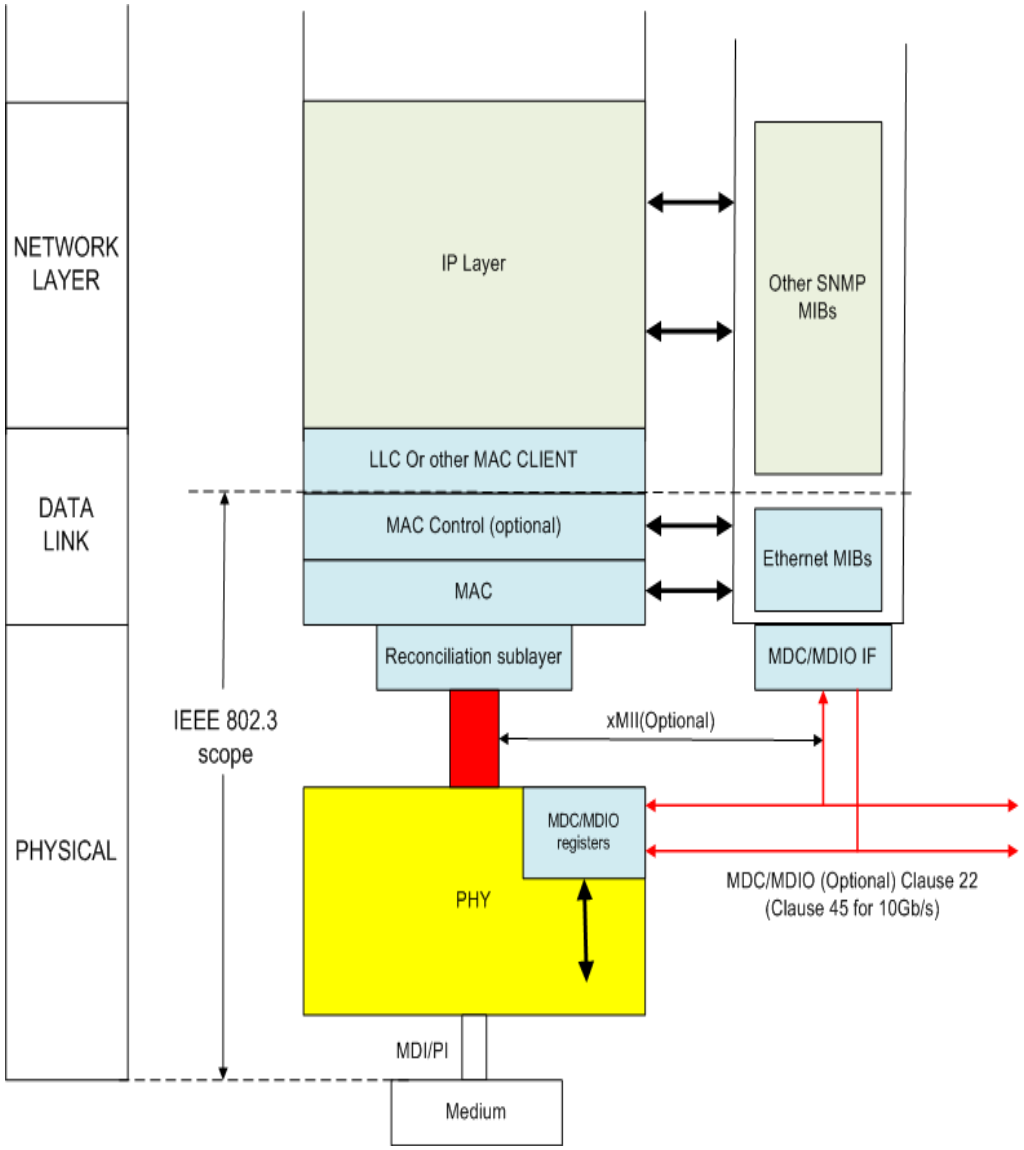


- IETF already defines a number of YANG models for different L3+ protocols
- 802.1 and 802.3 layers today modelled only with generic interface YANG models (very limited, no Ethernet specific content)

Section III

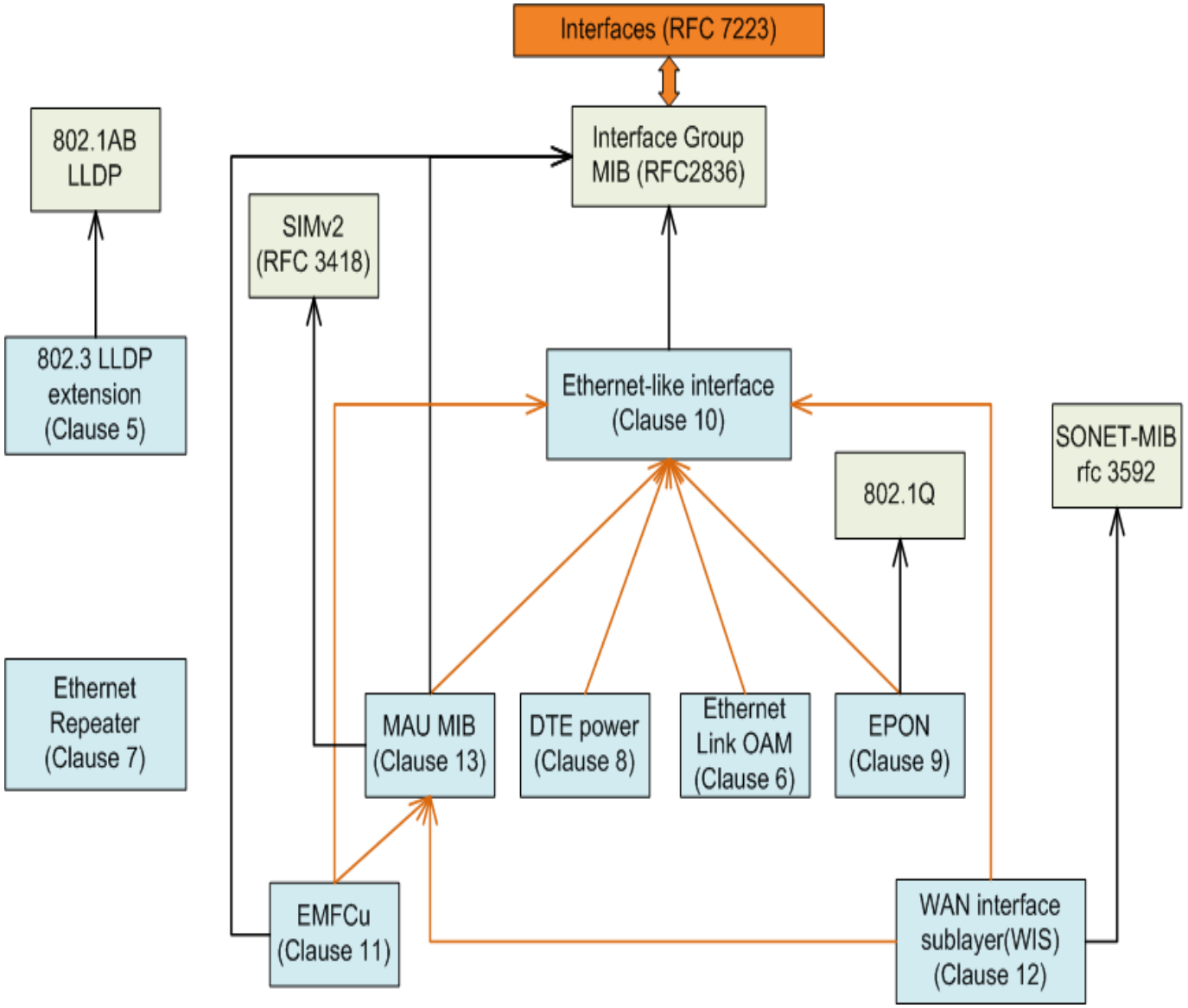
YANG FOR IEEE 802.3

IEEE 802.3 Ethernet Management

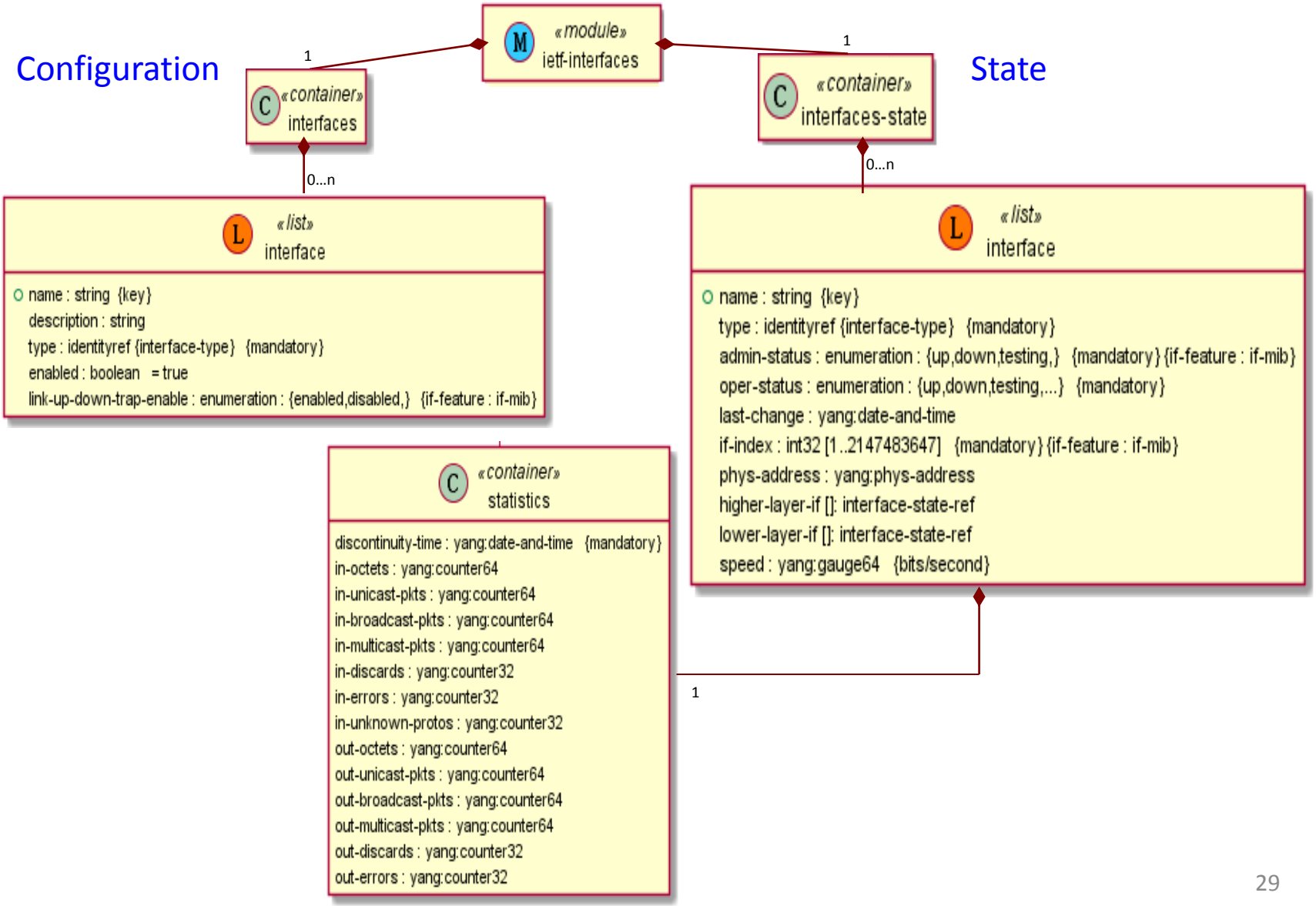


- Based on set of MIBs defined in IEEE Std 802.3.1
- Pervasive access to PHY via MDIO IF and mapping of registers into MIB objects
- Pervasive access to MAC / MAC Control and direct mapping into MIB objects
- Other MIBs (e.g., IETF MIB, 802.1 MIB, enterprise MIB) provide additional management functions, outside the scope of 802.3

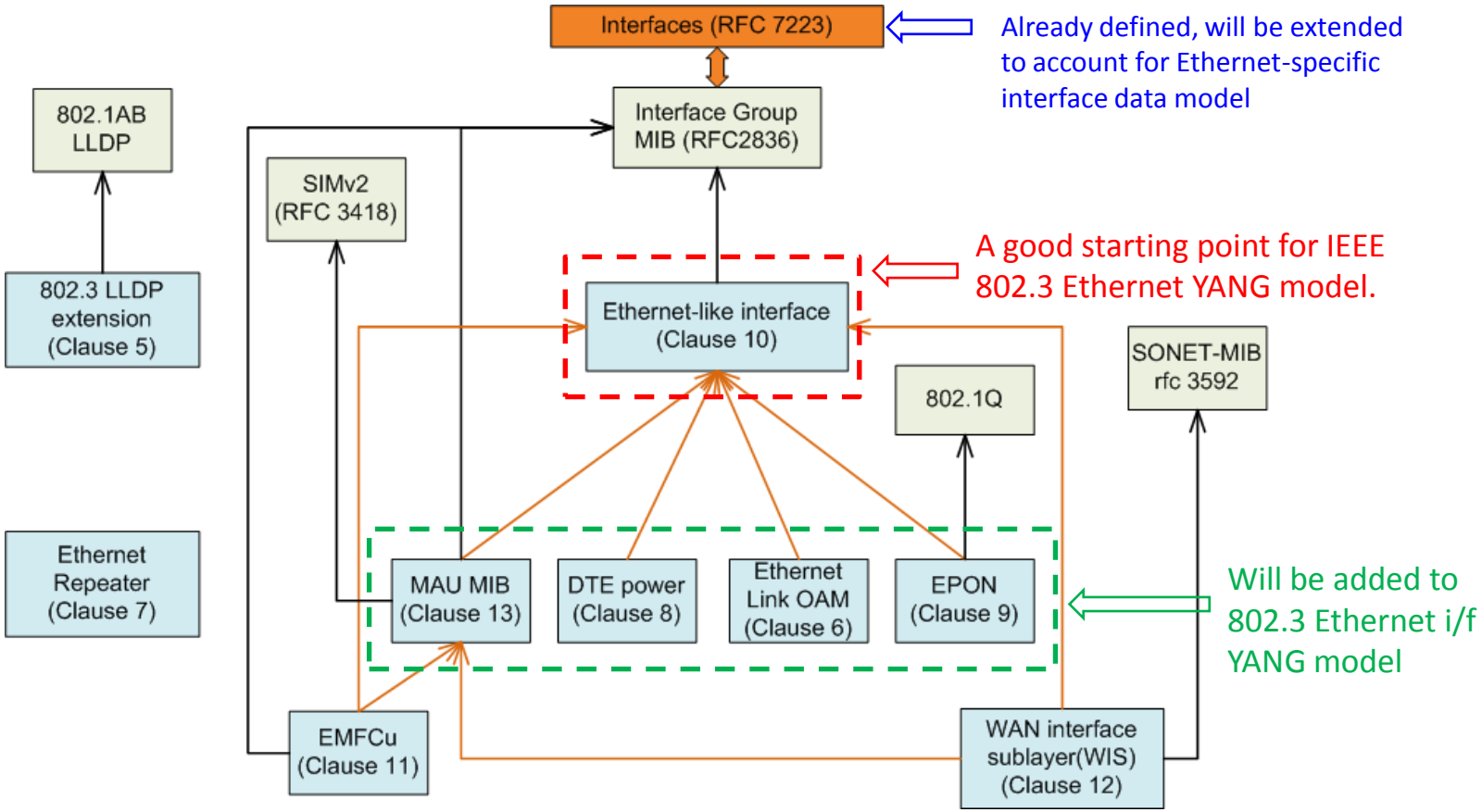
IEEE Std 802.3.1-2013 MIBs



IETF Interfaces Model (RFC 7223)



What do we need to do in 802.3?



- Extend existing RFC 7223 interface YANG model with Ethernet specific data
- Ethernet-like MIB in 802.3.1, C10 – the first to be "converted" to YANG
  - "conversion" implies cleanup and rationalization of existing MIB structures, and not just simple translation
- Other MIB, including EPON, DTE Power, EFM, etc. will extend basic Ethernet YANG model, once available

Scope of future project

- YANG model development in 802.3 will be part of larger undertaking in 802 as a whole
  - 802.1 is already working on their YANG models
  - 802.11 has some proprietary models in place already
- A small project (similar to 802.3.1) will be needed in 802.3 to “convert” existing 802.3.1 to YANG models and have them vetted by Ethernet community
  - Given simpler language syntax, no special knowledge of SMIv2/MIB is required
  - Focus on providing all necessary statistics, state information, and configuration hooks required by operators
  - This is not intended to be an exercise in translating 1:1 MIB ➔ YANG!

Why now?

- YANG+NETCONF/RESTCONF/… is the future management toolset for operators, providing much needed functionality and operational consistency across different vendors and implementations
- Development of proprietary Ethernet-like models is already under way, leading to interoperability problems
- IEEE 802.3 Working Group is *the* group responsible for development of Ethernet technology and *should* provide a standardized YANG model for the industry

Q&A?

End

Thanks