Network Configuration Management with NETCONF and YANG

Jürgen Schönwälder

84th IETF Meeting, Vancouver, 2012-07-29
See RFC 6632 for further details about the IETF’s contribution to the network management protocol soup.
NETCONF

The Network Configuration Protocol (NETCONF) provides mechanisms to install, manipulate, and delete the configuration of network devices. [RFC 6241]

YANG

YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF notifications. [RFC 6020]
1. Configuration Management [15 min]
2. NETCONF Protocol [20 min]
3. YANG Data Modeling Language [20 min]
4. Core Configuration Data Models [15 min]
5. Implementations, Tools, and Usage [10 min]
6. Future Directions [10 min]
7. Discussion [20 min]
Network operators typing / scripting proprietary CLIs

- Often coupled with a backup repository to track changes
- Labor intensive, expensive, error prone, widely deployed
“Generate Everything” (RFC 3139)

- Changes go to a network-wide configuration database
- Device configurations are automatically pushed to devices
- Devices are (ideally) never touched manually
R1 A configuration management protocol must be able to distinguish between configuration state and operational state. (configuration state vs. operational state)

- **Configuration state:**
  - Everything explicitly configured (e.g., IP addresses assigned manually to network interfaces)

- **Operational state:**
  - Usage and error counters obtained by the device
  - State learned from interaction with other devices (e.g., an IP address obtained from a DHCP server)
R2  A configuration management protocol must provide primitives to prevent errors due to concurrent configuration changes. (concurrency support)

- Goal: Prevent two operators from making configuration changes simultaneously that lead to undesired results
- Requires either strict locking or a conflict resolution mechanism
- Computer programs generally deal better with strict locking, conflict resolution requires intelligence
- Scope of locks determines how much concurrency is possible
R3 A configuration management protocol must provide primitives to apply configuration changes to a set of network elements in a robust and transaction-oriented way. (configuration transactions)

Example: renumbering a whole network

- All devices need to pick up new addresses
- During the renumbering process (activation of a new configuration), connectivity might be temporarily lost
- Devices need to be able to decide whether they keep the new configuration (addressing scheme) or rollback to the previous configuration
**R4** A configuration management protocol must be able to distinguish between several configurations and devices should be able to hold multiple configurations. (multiple configurations)

- Configurations can become large and complex
- Often useful to keep a few configurations on the device to easily switch between them (backup configs)
- Distinction between the running configuration and the configuration used at next re-initialization
R5 It is important to distinguish between the distribution of configurations and the activation of a certain configuration. (distribution vs. activation)

Example: peak hours vs. off-peak hours

- An operator wants to turn off links during off-peak hours
- This might happen once per day or even more frequently
- Instead of shipping a complete new configuration each time, it is much more efficient to activate a different already locally stored configuration

Implementations may internally keep data derived from known locally stored configurations in order to make the activation as smooth as possible
R6 A configuration management protocol must be clear about the persistence of configuration changes. (persistence of configuration state)

- Sometimes changes are temporary and should be forgotten after the next restart of a device
- Sometimes changes should be applied immediately and be remembered after reboots
- Sometimes changes should only be applied at the next restart of a device
R7 A configuration management protocol must be able to report configuration change events. (configuration change events)

- Example: firewall rules break applications
  - Customers report that certain applications suddenly fail to function correctly
  - Reason: A new firewall rule causes certain messages these applications rely on to be blocked

- A log of configuration change events often helps during fault isolation / resolution
R8 A full configuration dump and a full configuration restore are primitive operations frequently used by operators and must be supported appropriately. (configuration dump and restore)

- This seems to be rather obvious to operators
- Some network management protocols fail badly here
R9 A configuration management protocol must represent configuration state and operational state in a form enabling the use of existing tools for comparison, conversion, and versioning. (support for standard tools)

- Many operators use home-grown software systems
- Such systems often rely on standard tools for processing network management data
- Data formats enabling the use of off the shelf tools
  - eases integration
  - saves time
  - cuts costs
These requirements originate from an IAB workshop that paved the way for NETCONF and YANG [RFC3535]

<table>
<thead>
<tr>
<th>No</th>
<th>Description</th>
<th>SNMP</th>
<th>NETCONF</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>config vs. oper state</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>R2</td>
<td>concurrency support</td>
<td>o</td>
<td>+</td>
</tr>
<tr>
<td>R3</td>
<td>config transactions</td>
<td>-</td>
<td>[+</td>
</tr>
<tr>
<td>R4</td>
<td>multiple configs</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>R5</td>
<td>distribution vs. activation</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>R6</td>
<td>persistence of config state</td>
<td>o</td>
<td>+</td>
</tr>
<tr>
<td>R7</td>
<td>config change notifications</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>R8</td>
<td>config dump and restore</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>R9</td>
<td>support of standard tools</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Date</td>
<td>Milestone</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun 2002</td>
<td>IAB network management workshop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>May 2003</td>
<td>NETCONF working group established — work work work —</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dec 2006</td>
<td>NETCONF 1.0 RFCs published</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apr 2008</td>
<td>NETMOD working group established — work work work —</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oct 2010</td>
<td>YANG 1.0 RFCs published</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jun 2011</td>
<td>NETCONF 1.1 RFCs published</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4 2012</td>
<td>Core configuration data models published</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Configuration Management [15 min]
2. NETCONF Protocol [20 min]
3. YANG Data Modeling Language [20 min]
4. Core Configuration Data Models [15 min]
5. Implementations, Tools, and Usage [10 min]
6. Future Directions [10 min]
7. Discussion [20 min]
## NETCONF RFCs (as of today)

<table>
<thead>
<tr>
<th>RFC</th>
<th>Status</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC 6241</td>
<td>PS</td>
<td>NETCONF Protocol Version 1.1</td>
</tr>
<tr>
<td>RFC 6242</td>
<td>PS</td>
<td>NETCONF over SSH Version 1.1</td>
</tr>
<tr>
<td>RFC 6243</td>
<td>PS</td>
<td>NETCONF With-defaults Capability</td>
</tr>
<tr>
<td>RFC 6470</td>
<td>PS</td>
<td>NETCONF Base Notifications</td>
</tr>
<tr>
<td>RFC 6536</td>
<td>PS</td>
<td>NETCONF Access Control Model</td>
</tr>
<tr>
<td>RFC 5717</td>
<td>PS</td>
<td>NETCONF Partial Locking</td>
</tr>
<tr>
<td>RFC 5277</td>
<td>PS</td>
<td>NETCONF Event Notifications</td>
</tr>
<tr>
<td>RFC 6022</td>
<td>PS</td>
<td>NETCONF Monitoring</td>
</tr>
<tr>
<td>RFC 6244</td>
<td>I</td>
<td>NETCONF / YANG Architecture</td>
</tr>
</tbody>
</table>

- Further RFCs define additional transports
- Some of the transports may be declared historic soon
NETCONF enabled devices include a NETCONF server
Management applications include a NETCONF client
Device CLIs can be wrapped around a NETCONF client
- Security is provided by the transport layer
- Operations focus on primitives to manage configurations
- The set of operations is designed to be extensible
A configuration datastore is the complete set of configuration information that is required to get a device from its initial default state into a desired operational state.

- The `<running>` configuration datastore represents the currently active configuration of a device and is always present.
- The `<startup>` configuration datastore represents the configuration that will be used during the next startup.
- The `<candidate>` configuration datastore represents a configuration that may become a `<running>` configuration through an explicit commit.
Some operations (edit-config) may support different selectable error behaviours, including rollback behaviour.
After establishing a session over a secure transport, both NETCONF protocol engines send a hello message to announce their protocol capabilities, the supported data models, and the server’s session identifier.

```
S: <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
S:   <capabilities>
S:     <capability>
S:     </capability>
S:     <capability>
S:     </capability>
S:     <capability>
S:         module=ietf-interfaces&revision=2012-04-29
S:     </capability>
S:   </capabilities>
S:   <session-id>4</session-id>
S: </hello>
```
The Remote Procedure Call (RPC) protocol consists of a `<rpc/>` message followed by an `<rpc-reply/>` message.

C: `<rpc message-id="101">
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-config>
   <source>
   <running/>
   </source>
   </get-config>
</rpc>

S: `<rpc-reply message-id="101">
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <data><!-- ...contents here... --></data>
</rpc-reply>`
RPC failures are indicated by one or more `<rpc-error/>` elements contained in the `<rpc-reply/>` element.

C: `<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
C:   <get-config><source><running/></source></get-config>
C: </rpc>

S: `<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
S:   <rpc-error>
S:     <error-type>rpc</error-type>
S:     <error-tag>missing-attribute</error-tag>
S:     <error-severity>error</error-severity>
S:     <error-info>
S:       <bad-attribute>message-id</bad-attribute>
S:       <bad-element>rpc</bad-element>
S:     </error-info>
S:   </rpc-error>
S: </rpc-reply>`
NETCONF Operations (RFC6241)

- get-config(source, filter) $\rightarrow$ data
  Retrieve all or part of the configuration datastore source.

- edit-config(target, default-operation, test-option, error-option, config)
  Edit the configuration datastore target by merging, replacing, creating, or deleting new config elements. The test-option parameter allows to do a “dry run” while the error-option parameter controls how the server reacts to errors (stop, continue, rollback).

- copy-config(target, source)
  Copy the content of the configuration datastore source to the configuration datastore target.
NETCONF Operations (RFC6241)

- `delete-config(target)`
  Delete the named configuration datastore target.

- `lock(target)`
  Lock the configuration datastore target.

- `unlock(target)`
  Unlock the configuration datastore target.

- `validate(source)`
  Validate the configuration datastore source (:validate capability).
NETCONF Operations (RFC6241)

- **get(filter) → data**
  Retrieve all or part of the running configuration data store and merged with the device’s state information.

- **close-session()**
  Gracefully close the current session.

- **kill-session(session-id)**
  Force the termination of the session session-id.
▷ discard-changes()

Revert the candidate configuration datastore to the running configuration (:candidate capability).

▷ commit(confirmed, confirm-timeout, persist, persist-id)

Commit the candidate to the running configuration datastore with an optional automatic rollback (:candidate, :confirmed-commit capability).

▷ cancel-commit(persist-id)

Cancel an ongoing confirmed commit identified by persist-id (:confirmed-commit capability).
More NETCONF Operations (RFC5717, RFC5277)

- partial-lock(select)
  Create a partial lock on the running configuration datastore on the nodes / subtree identified by the select XPath expression (RFC 5717).

- partial-unlock(lock-id)
  Remove a previously established partial lock (RFC 5717).

- create-subscription(stream, filter, start, stop)
  Subscribe to a notification stream using an optional filter; the optional start/stop times support the replay of notifications (RFC 5277).
Embedded operation attributes specify how a configuration is modified by an edit-config() configuration.

- **merge:**
  Configuration data is merged with the configuration.

- **replace:**
  Configuration data replaces existing configuration.

- **create:**
  Configuration data is added iff it does not already exist.

- **delete:**
  Configuration data is deleted from the datastore.
```xml
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface nc:operation="replace">
          <name>Ethernet0/0</name>
          <mtu>1500</mtu>
        </interface>
      </interfaces>
    </config>
  </edit-config>
</rpc>
```
Subtree filter expressions select XML subtrees to include in get and get-config responses (XPath expressions can be used as well if the :xpath capability is supported).

- namespace selection (wildcarding)
  A non-null namespace of a node in a subtree filter matches the node in the given namespace. A null namespace acts as a wildcard.

- attribute match expressions
  The set of XML attributes present in any type of filter node form an “attribute match expression”; the selected data must have matching values for every attribute of an attribute match expression.
subtree filtering (cont.)

- **containment nodes**
  
  For each containment node of subtree filter, all data model instances must exactly match the specified namespaces, element hierarchy, and any attribute match expressions.

- **selection nodes**
  
  An empty leaf node of a filter selects the specified subtree(s) and it suppresses the automatic selection of the entire set of sibling nodes.

- **content match nodes**
  
  A leaf node of a filter containing simple content selects some or all of its sibling nodes.
<filter type="subtree">

<!-- namespace selection and containment node selection -->
<t:top xmlns:t="http://example.com/schema/1.2/config">

<!-- containment node selection -->
<t:interfaces>

<!-- containment node selection and attribute match expression -->
<!-- (note that YANG does not use XML attributes) -->
<t:interface t:ifName="eth0">

<!-- selection node -->
<t:ifSpeed/>

<!-- content match node -->
<t:type>ethernet</t:if-type>

</t:interface>
</t:interfaces>
</t:top>
</filter>
Motivation: Use an already widely deployed security protocol for CLIs, reducing the costs associated with key management.

- SSH supports multiple logical channels over one transport layer association

- For framing purposes, the special end of message marker `]]>]]>` is used for the initial `hello` message

- Subsequent NETCONF 1.1 messages use a chunked framing format (if both parties use NETCONF 1.1)

- SSH is the mandatory to implement NETCONF transport
Server announces protocol version, startup capability, and the ietf-interfaces data model
C: <?xml version="1.0" encoding="UTF-8"?>
C: <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
C:    <capabilities>
C:      <capability>
C:        urn:ietf:params:xml:ns:netconf:base:1.1
C:      </capability>
C:    </capabilities>
C: </hello>
C: ]]>]]>

- Client announces protocol version
- Client and server switch to chunked framing (both announced support for NETCONF 1.1)
NETCONF over SSH: Chunked Framing

C: #350
C: <rpc message-id="105"
C: xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
C:   <get-config>
C:     <source>
C:       <running/>
C:     </source>
C:   <filter type="subtree">
C:     <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
C:       <interface>
C:         <type>ethernetCsmacd</type>
C:       </interface>
C:     </interfaces>
C:   </filter>
C: </get-config>
C: </rpc>
C: ##

Messages are send in chunks (limiting buffer sizes)
Each chunk is prefixed by its length
A special marker indicates the end of a message
S: #165
S: <rpc-reply message-id="105">
S:     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
S:     <data>
S:         <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
S:             <interface>
S:                 <name>eth0</name><type>ethernetCsmacd</type>
S:                 <location>0</location><enabled>true</enabled>
S:                 <if-index>2</if-index>
S:             </interface>
S:             <interface>
S:                 <name>eth1</name><type>ethernetCsmacd</type>
S:                 <location>1</location><enabled>false</enabled>
S:                 <if-index>7</if-index>
S:             </interface>
S:         </interfaces>
S:     </data>
S: </rpc-reply>
S: ##
1. Configuration Management [15 min]
2. NETCONF Protocol [20 min]
3. YANG Data Modeling Language [20 min]
4. Core Configuration Data Models [15 min]
5. Implementations, Tools, and Usage [10 min]
6. Future Directions [10 min]
7. Discussion [20 min]
### YANG RFCs (as of today)

<table>
<thead>
<tr>
<th>RFC</th>
<th>Status</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC 6020</td>
<td>PS</td>
<td>YANG Version 1</td>
</tr>
<tr>
<td>RFC 6021</td>
<td>PS</td>
<td>Common Data Types</td>
</tr>
<tr>
<td>RFC 6087</td>
<td>PS</td>
<td>Guidelines for Authors/Reviewers</td>
</tr>
<tr>
<td>RFC 6095</td>
<td>E</td>
<td>YANG Language Abstractions</td>
</tr>
<tr>
<td>RFC 6110</td>
<td>PS</td>
<td>Mapping YANG to DSDL</td>
</tr>
<tr>
<td>RFC 6643</td>
<td>PS</td>
<td>Mapping SMIv2 to YANG</td>
</tr>
<tr>
<td>RFC 6244</td>
<td>I</td>
<td>NETCONF / YANG Architecture</td>
</tr>
</tbody>
</table>

- YANG data models can be found in RFC 6022, RFC 6241, RFC 6243, RFC 6470, RFC 6536, ... 
- We expect this list to grow soon with your help
YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol, NETCONF operations, and NETCONF notifications.

- YANG uses a compact syntax since human readability is highest priority
- YIN is an XML representation of YANG (lossless roundtrip conversion)
- YANG can be translated to XML Schema and RELAX NG so that existing XML tools can be utilized
- YANG can be translated to Schematron to validate NETCONF content
A module is a self-contained collection of YANG definitions

A submodule is a partial module definition which contributes derived types, groupings, data nodes, RPCs, and notifications to a module
module ietf-inet-types {

    namespace "urn:ietf:params:xml:ns:yang:ietf-inet-types";
    prefix "inet";

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

    contact
        "WG Web: <http://tools.ietf.org/wg/netmod/>";

    description
        "This module contains a collection of generally useful derived
         YANG data types for Internet addresses and related things.";

    revision 2010-09-24 {
        description
            "Initial revision.";
        reference
            "RFC 6021: Common YANG Data Types";
    }
}
The data type system is mostly an extension of the SMIng type system, accommodating XML and XSD requirements.

- Compatible with the XSD / RelaxNG type systems
module ietf-inet-types {

  namespace "urn:ietf:params:xml:ns:yang:ietf-inet-types";
  prefix "inet";

  typedef ipv4-address {
    type string {
      pattern '(((\[0-1]?[0-9]?(0-9)|2[0-4]\[0-9]|25[0-5])\.)\{3\}'
        + '([0-1]?[0-9]?(0-9)|2[0-4]\[0-9]|25[0-5])'
        + '(%[\p{N}\p{L}]\{\?p\{N\}\p{L}\}\{\})\)?';
    }
  }

  typedef ip-address {
    type union {
      type inet:ipv4-address;
      type inet:ipv6-address;
    }
    description
      "The ip-address type represents an IP address and is IP
      version neutral. The format of the textual representations
      implies the IP version.";
  }
}
- **Leaf:**
  A leaf has one value, no children, one instance.

- **Leaf-list:**
  A leaf-list has one value, no children, multiple instances.

- **Container:**
  A container has no value, holds related children, has one instance.

- **List:**
  A list has no value, holds related children, has multiple instances, has a key property.
Example: leaf and leaf-list

```yaml
leaf domain {
    type inet:domain-name; // values are typed (type imported)
    mandatory true;       // must exist in a valid configuration
    config true;         // part of the set of configuration objects
    description
        "The host name of this system."
}

// XML: <domain>example.com</domain>

leaf-list search {
    type inet:domain-name; // imported from the module with prefix inet
    ordered-by user;      // maintain the order given by the user
    description
        "List of domain names to search."
}

// XML: <search>eng.example.com</search>
// XML: <search>example.com</search>
```
Example: container

```
container system {
    config true;
    leaf hostname {
        type inet:domain-name;
    }
    container resolver {
        leaf domain { /* see above */ }
        leaf-list search { /* see above */ }
        description
            "The configuration of the resolver library."
    }
}

// XML: <system>
// XML:  <hostname>server.example.com</hostname>
// XML:  <resolver>
// XML:   <domain>example.com</domain>
// XML:   <search>eng.example.com</search>
// XML:   <search>example.com</search>
// XML:  </resolver>
// XML: </system>
```
list nameserver {
    key address;
    leaf address {
        type inet:ip-address;
    }
    leaf status {
        type enumeration {
            enum enabled; enum disabled;
        }
    }
}

// XML: <nameserver>
// XML: <address>192.0.2.1</address>
// XML: <status>enabled</status>
// XML: </nameserver>

// XML: <nameserver>
// XML: <address>192.0.2.2</address>
// XML: <status>disabled</status>
// XML: </nameserver>
Augment, Must, When, Presence

▶ augment:
The augment statement places nodes into an existing hierarchy using the current module’s namespace.

▶ must:
The must statement expresses constraints (XPath expressions) that must be satisfied by a valid configuration.

▶ when:
The when statement can be used to define sparse augmentations where nodes are only added when a condition (XPATH expression) is true.

▶ presence:
The existence of a presence container carries a certain meaning (a single bit of configuration data).
Example: augment and presence

```
augment /system/resolver {
    container debug {
        presence "enables debugging";
        description
            "This container enables debugging.";
        leaf level {
            type enumeration {
                enum low; enum medium; enum full;
            }
            default "medium";
            mandatory false;
            description
                "The debugging level; default is medium debug information.";
        }
    }
}

// XML: <system><resolver>
// XML:   <debug/>
// XML: </resolver></system>
```
Example: augment and must

```go
augment /system/resolver {
  leaf access-timeout {
    type uint32;
    unit "seconds";
    mandatory true;
    description "Maximum time without server response.";
  }
  leaf retry-timer {
    type uint32;
    units "seconds";
    description "Period after which to retry an operation";
    must ". < ../access-timeout" {
      error-app-tag "retry-timer-invalid";
      error-message "The retry timer must be less "
          + "than the access timeout";
    }
  }
}
```
Example: augment and when

```python
augment /system/resolver/nameserver {
    when "status = 'enabled'";
    leaf tx {
        type yang:counter32;
        config false;
    }
    leaf rx {
        type yang:counter32;
        config false;
    }
}
```

// XML:  <nameserver>
// XML:   <address>192.0.2.1</address>
// XML:   <status>enabled</status>
// XML:   <tx>2345</tx>
// XML:   <rx>1234</rx>
// XML:  </nameserver>
// XML:  <nameserver>
// XML:   <address>192.0.2.2</address>
// XML:   <status>disabled</status>
// XML:  </nameserver>
```
grouping:
A grouping is a reusable collection of nodes and it can be used to emulate structured data types. A grouping can be refined when it is used.

choice:
A choice allows one alternative of the choice to exist. It can be used to provide extensibility hooks to be exploited using augments.

notification:
The notification statement can be used to define the contents of event notifications.

rpc:
The rpc statement can be used to define operations and their input and output parameters.
Example: grouping

grouping target {
    leaf address {
        type inet:ip-address;
        description "Target IP address."
    }

    leaf port {
        type inet:ip-port;
        description "Target port number."
    }
}

list nameserver {
    key "address port";
    uses target;
}

// XML: <nameserver>
// XML:   <address>192.0.2.1</address>
// XML:   <port>53</port>
// XML:   </nameserver>
Example: choice

container transfer {
    choice how {
        default interval;
        case interval {
            leaf interval {
                type uint16; default 30; units minutes;
            }
        }
        case daily {
            leaf daily {
                type empty;
            }
            leaf time-of-day {
                type string; units 24-hour-clock; default 1am;
            }
        }
        case manual {
            leaf manual {
                type empty;
            }
        }
    }
}
notification nameserver-failure {
    description
        "A failure of a nameserver has been detected and the server has been disabled."
    leaf address {
        type leafref {
            path "/system/resolver/nameserver/address";
        }
    }
}

// MSG: <notification>
// MSG: <eventTime>2008-06-03T18:34:50+02:00</eventTime>
// MSG: <nameserver-failure>
// MSG: <address>192.0.2.2</address>
// MSG: </nameserver-failure>
// MSG: </notification>
rpc activate-software-image {
    input {
        leaf image-name {
            type string;
        }
    }
    output {
        leaf status {
            type string;
        }
    }
}

// RPC: <rpc message-id="42">
// RPC:  <activate-software-image xmlns="urn:mumble">
// RPC:       <image-name>image.tgz</image-name>
// RPC:    </activate-software-image>
// RPC: </rpc>
1. Configuration Management [15 min]
2. NETCONF Protocol [20 min]
3. YANG Data Modeling Language [20 min]
4. Core Configuration Data Models [15 min]
5. Implementations, Tools, and Usage [10 min]
6. Future Directions [10 min]
7. Discussion [20 min]
Core Interfaces Data Model

A technology agnostic model of network interfaces

To be augmented with interface type specific nodes

Interfaces can be layered 1:N and N:1

Configuration of layering is technology specific
IPv4 interface configuration augments interfaces

- The (subnet)? line indicates a choice and the (prefix-length) and (netmask) lines indicate a case

- Both, choice and case do not appear in the config
IPv6 interface configuration similarly augments interfaces
The main routing table is always present, additional routing tables can be configured.

Route filters control the propagation of routes.

Core data model plus IP4 and IPv6 unicast routing models.
---rw routing
  +--rw router [name]
    |   +--rw name
    |   +--rw router-id?
    |   +--rw description?
    |   +--rw enabled?
    |   +--rw interfaces
    |     |   +--rw interface [name]
    |     |     +--rw name
    |     +--rw routing-protocols
    |         +--rw routing-protocol [name]
    |         |   +--rw name
    |         |   +--rw description?
    |         |   +--rw type
    |         |   +--rw connected-routing-tables
    |         |     |   +--rw routing-table [name]
    |         |     |     +--rw name
    |         |     |     +--rw import-filter?
    |         |     |     +--rw export-filter?
    |         |   +--rw static-routes
Core IP Routing Data Model (ietf-routing)

| +--rw routing-tables
|   +--rw routing-table [name]
|     +--rw name
|     +--rw address-family?
|     +--rw safi?
|     +--rw description?
|     +--ro routes
|     |   +--ro route
|     |     +--ro source-protocol
|     |     +--ro age
|     +--rw recipient-routing-tables
|     |   +--rw recipient-routing-table [name]
|     |     +--rw name
|     |     +--rw filter?
|   +--rw route-filters
|     +--rw route-filter [name]
|         +--rw name
|         +--rw description?
|         +--rw type?
Core System Data Model: System Identification

rpcs:
  +---x set-current-datetime
  |   +--ro input
  |   |   ++--ro current-datetime yang:date-and-time
  +---x system-restart
  +---x system-shutdown

---rw system
  +---rw contact? string
  +---rw name? string
  +---rw location? string
  +---ro platform
  |   ++--ro os-name? string
  |   |   ++--ro os-release? string
  |   |   |   ++--ro os-version? string
  |   |   |   |   ++--ro machine? string
  |   |   |   |   |   ++--ro nodename? string

▷ Basic information about a device
▷ Operations to restart/shutdown as device
Information about current and boot time

Timezone configuration

Basic NTP client configuration
+-rw dns
|  +-rw search*       inet:host
|  +-rw server*       inet:ip-address
|  +-rw options
|   |  +-rw ndots?       uint8
|   |  +-rw timeout?      uint8
|   |  +-rw attempts?     uint8
+-rw radius
|  +-rw server [address]
|   |   +-rw address       inet:host
|   |   +-rw authentication-port? inet:port-number
|   |   +-rw shared-secret? string
|  +-rw options
|   |  +-rw timeout?      uint8
|   |  +-rw attempts?     uint8
+-rw authentication
   +-rw user-authentication-order* identityref
   +-rw user [name]
      |  +-rw name         string
      |  +-rw password?    crypt-hash
      |  +-rw ssh-dsa?     binary
      |  +-rw ssh-rsa?     binary
## Other Yang Data Models

<table>
<thead>
<tr>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>NETCONF Monitoring Data Model</td>
<td>RFC6022</td>
</tr>
<tr>
<td>NETCONF Access Control Data Model</td>
<td>RFC6536</td>
</tr>
<tr>
<td>IPFIX Configuration Data Model</td>
<td>RFC Editor</td>
</tr>
<tr>
<td>SNMP Configuration Data Model</td>
<td>WG Draft</td>
</tr>
</tbody>
</table>

- The groundwork has been done
- Time to build data models on top of it
- Should ideally be done by subject matter experts
- YANG experts are happy to assist and review
Implementations, Tools, and Usage

1. Configuration Management [15 min]
2. NETCONF Protocol [20 min]
3. YANG Data Modeling Language [20 min]
4. Core Configuration Data Models [15 min]
5. Implementations, Tools, and Usage [10 min]
6. Future Directions [10 min]
7. Discussion [20 min]
# NETCONF Implementations

## Commercial (not necessarily complete)

<table>
<thead>
<tr>
<th>Company</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applied Informatics</td>
<td><a href="http://www.appinf.com/">http://www.appinf.com/</a></td>
</tr>
<tr>
<td>Centered Logic</td>
<td><a href="http://www.centeredlogic.com/">http://www.centeredlogic.com/</a></td>
</tr>
<tr>
<td>MG-Soft</td>
<td><a href="http://www.mg-soft.si/">http://www.mg-soft.si/</a></td>
</tr>
<tr>
<td>Oracle</td>
<td><a href="http://www.oracle.com/">http://www.oracle.com/</a></td>
</tr>
<tr>
<td>Tail-f</td>
<td><a href="http://www.tail-f.com/">http://www.tail-f.com/</a></td>
</tr>
<tr>
<td>WebNMS</td>
<td><a href="http://www.webnms.com/">http://www.webnms.com/</a></td>
</tr>
<tr>
<td>YumaPro</td>
<td><a href="http://www.yumaworks.com/">http://www.yumaworks.com/</a></td>
</tr>
</tbody>
</table>

## Open Source (not necessarily complete)

<table>
<thead>
<tr>
<th>Project</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnSuite</td>
<td><a href="http://ensuite.sourceforge.net/">http://ensuite.sourceforge.net/</a></td>
</tr>
<tr>
<td>ncclient</td>
<td><a href="http://code.google.com/p/ncclient/">http://code.google.com/p/ncclient/</a></td>
</tr>
<tr>
<td>netconfx</td>
<td><a href="http://www.centeredlogic.com/">http://www.centeredlogic.com/</a></td>
</tr>
<tr>
<td>netconf4j</td>
<td><a href="https://github.com/dana-i2cat/netconf4j">https://github.com/dana-i2cat/netconf4j</a></td>
</tr>
<tr>
<td>netconf4android</td>
<td><a href="http://code.google.com/p/netconf4android/">http://code.google.com/p/netconf4android/</a></td>
</tr>
<tr>
<td>netopeer</td>
<td><a href="http://code.google.com/p/netopeer/">http://code.google.com/p/netopeer/</a></td>
</tr>
<tr>
<td>Yuma</td>
<td><a href="http://sourceforge.net/projects/yuma/">http://sourceforge.net/projects/yuma/</a></td>
</tr>
<tr>
<td>Device Vendor</td>
<td>Website</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>Alaxala</td>
<td><a href="http://www.alaxala.com/">http://www.alaxala.com/</a></td>
</tr>
<tr>
<td>Telco Systems</td>
<td><a href="http://www.telco.com/">http://www.telco.com/</a></td>
</tr>
<tr>
<td>BigBand/Arris</td>
<td><a href="http://www.arrisi.com/">http://www.arrisi.com/</a></td>
</tr>
<tr>
<td>Brocade</td>
<td><a href="http://www.brocade.com/">http://www.brocade.com/</a></td>
</tr>
<tr>
<td>Cisco Systems</td>
<td><a href="http://www.cisco.com/">http://www.cisco.com/</a></td>
</tr>
<tr>
<td>Edgeware</td>
<td><a href="http://www.edgeware.tv/">http://www.edgeware.tv/</a></td>
</tr>
<tr>
<td>Ericsson</td>
<td><a href="http://www.ericsson.com/">http://www.ericsson.com/</a></td>
</tr>
<tr>
<td>H3C</td>
<td><a href="http://www.h3c.com/">http://www.h3c.com/</a></td>
</tr>
<tr>
<td>Huawei</td>
<td><a href="http://www.huawei.com/">http://www.huawei.com/</a></td>
</tr>
<tr>
<td>Juniper Networks</td>
<td><a href="http://www.juniper.net/">http://www.juniper.net/</a></td>
</tr>
<tr>
<td>Nexor</td>
<td><a href="http://www.nexor.com/">http://www.nexor.com/</a></td>
</tr>
<tr>
<td>RuggedCom</td>
<td><a href="http://www.ruggedcom.com/">http://www.ruggedcom.com/</a></td>
</tr>
<tr>
<td>Sonus</td>
<td><a href="http://www.sonus.net/">http://www.sonus.net/</a></td>
</tr>
<tr>
<td>Taseon</td>
<td><a href="http://www.taseon.com/">http://www.taseon.com/</a></td>
</tr>
<tr>
<td>Verivue</td>
<td><a href="http://www.verivue.com/">http://www.verivue.com/</a></td>
</tr>
</tbody>
</table>
## YANG Implementations

<table>
<thead>
<tr>
<th>Commercial (not necessarily complete)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MG-Soft</td>
<td><a href="http://www.mg-soft.si/">http://www.mg-soft.si/</a></td>
</tr>
<tr>
<td>Tail-f (ConfD)</td>
<td><a href="http://www.tail-f.com/">http://www.tail-f.com/</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Open Source (not necessarily complete)</th>
<th></th>
</tr>
</thead>
</table>
### IETF (not necessarily complete)

<table>
<thead>
<tr>
<th>Organization</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPFIX</td>
<td>IPFIX Configuration Model</td>
</tr>
<tr>
<td>NETMOD</td>
<td>Interfaces Core Configuration Model</td>
</tr>
<tr>
<td>NETMOD</td>
<td>IP and Routing Core Data Models</td>
</tr>
<tr>
<td>NETMOD</td>
<td>System Configuration Data Model</td>
</tr>
<tr>
<td>NETMOD</td>
<td>SNMP Configuration Data Model</td>
</tr>
<tr>
<td>NETCONF</td>
<td>NETCONF Monitoring Data Model</td>
</tr>
<tr>
<td>NETCONF</td>
<td>NETCONF Access Control Data Model</td>
</tr>
</tbody>
</table>

### Other SDOs (not necessarily complete)

<table>
<thead>
<tr>
<th>Organization</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Networking Found.</td>
<td>OF-Config 1.1</td>
</tr>
<tr>
<td>CableLabs</td>
<td>CM-SP-CCAP-OSSI-I02-120329</td>
</tr>
<tr>
<td>Metro Ethernet Forum</td>
<td>PM and FM (to be published)</td>
</tr>
</tbody>
</table>
YANG Mapping to JSON

The mapping defines a procedure for translating the subset of YANG-compatible XML documents to JSON text. The translation is driven by a YANG data model which must therefore be known in advance.

- JSON is a popular compact and easy to parse data format used by many REST APIs
- Translation of YANG namespaces is supported
- YANG datatype information is used to translate leaf values to the most appropriate JSON representation
- Slightly more compact (irrelevant with compression)
- Increased human readability (less noise)
YANG to JSON Example

```json
{
    "ietf-interfaces:interfaces": {
        "interface": [
            {
                "name": "eth0",
                "type": "ethernetCsmacd",
                "location": "0",
                "enabled": true,
                "if-index": 2
            },
            {
                "name": "eth1",
                "type": "ethernetCsmacd",
                "location": "1",
                "enabled": false,
                "if-index": 2
            }
        ]
    }
}
```
```
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
  <interface>
    <name>eth0</name>
    <type>ethernetCsmacd</type>
    <location>0</location>
    <enabled>true</enabled>
    <if-index>2</if-index>
  </interface>
  <interface>
    <name>eth1</name>
    <type>ethernetCsmacd</type>
    <location>1</location>
    <enabled>false</enabled>
    <if-index>7</if-index>
  </interface>
</interfaces>
```

- 214 octets in JSON format (all white space removed)
- 347 octets in XML format (all white space removed)
A RESTful protocol that provides a programmatic interface over HTTP for accessing data defined in YANG, using the datastores defined in NETCONF.

- Configuration data and state data are exposed as resources that can be retrieved with the GET method.
- Resources representing configuration data can be modified with the DELETE, PATCH, POST, and PUT methods.
- Data-model specific RPC operations defined with the YANG “rpc” statement can be invoked with the POST method.
- Optional transaction resource is used to allow batching of edits and handling of concurrent editing transactions.
C: GET /yang-api HTTP/1.1
C: Host: example.com

S: HTTP/1.1 200 OK
S: Date: Mon, 23 Apr 2012 17:01:00 GMT
S: Server: example-server
S: Content-Type: application/vnd.yang.api+json
S:
S: {
S:  "yang-api": {
S:   "capabilities": {
S:     "edit-model": "direct",
S:     "persist-model": "automatic",
S:     "transaction-model": "none"
S:   },
S:   "modules": {
S:     "module": [
S:     ]
S:   },
S:   "version": "1.0"
S:  }
S: }
S: }
C: POST /yang-api/datastore/jukebox HTTP/1.1
C: Host: example.com

S: HTTP/1.1 201 Created
S: Date: Mon, 23 Apr 2012 17:01:00 GMT
S: Server: example-server
S: Location: http://example.com/yang-api/datastore/jukebox
S: Last-Modified: Mon, 23 Apr 2012 17:01:00 GMT
S: ETag: b3a3e673be2

C: POST /yang-api/datastore/jukebox/artist HTTP/1.1
C: Host: example.com
C: Content-Type: application/vnd.yang.data+json
C:
C: { "artist" : { "name" : "The Foo Fighters" } }

S: HTTP/1.1 201 Created
S: Date: Mon, 23 Apr 2012 17:02:00 GMT
S: Server: example-server
S: Location: http://example.com/yang-api/datastore/jukebox/artist/1
S: Last-Modified: Mon, 23 Apr 2012 17:02:00 GMT
S: ETag: b3830f23a4c
Discussion

1. Configuration Management [15 min]
2. NETCONF Protocol [20 min]
3. YANG Data Modeling Language [20 min]
4. Core Configuration Data Models [15 min]
5. Implementations, Tools, and Usage [10 min]
6. Future Directions [10 min]
7. Discussion [20 min]
Summary and Advice...

- If you are working on configuration of something, consider NETCONF seriously before rolling your own protocol.
- NETCONF is based on years of experience.
- NETCONF over BEEP or SOAP may disappear.
- There are solid (open source) implementations.

- Writing data models in YANG can actually be fun.
- There are very good (open source) tools around.
- Review existing YANG modules to see how your model fits into the core set of data models being defined.
- Check RFC 6087: Guidelines for Authors and Reviewers.
- Ask YANG doctors for help as needed.


Mapping YANG to Document Schema Definition Languages and Validating NETCONF Content.
RFC 6110, CESNET, February 2011.

Translation of Structure of Management Information Version 2 (SMIpv2) MIB Modules to YANG Modules.
RFC 6643, Jacobs University, July 2012.

Extending YANG with Language Abstractions.

Network Configuration Protocol (NETCONF).
RFC 6241, Juniper Networks, Tail-f Systems, Jacobs University, Brocade, June 2011.

Using the NETCONF Protocol over Secure Shell (SSH).

With-defaults Capability for NETCONF.
RFC 6243, Brocade, Ericsson, June 2011.

Network Configuration Protocol (NETCONF) Base Notifications.
RFC 6470, Brocade, February 2012.

Network Configuration Protocol (NETCONF) Access Control Model.
NETCONF Event Notifications.  

YANG Module for NETCONF Monitoring.  
RFC 6022, Ericsson, Tail-f Systems, October 2010.

Partial Lock Remote Procedure Call (RPC) for NETCONF.  

A YANG Data Model for Interface Configuration.  

A YANG Data Model for IP Configuration.  

[22] L. Lhotka.  
A YANG Data Model for Routing Configuration.  
Internet-Draft (work in progress) <draft-ietf-netmod-routing-cfg-04>, CZ.NIC, July 2012.

[23] A. Bierman and M. Bjorklund.  
YANG Data Model for System Management.  

Modeling JSON Text with YANG.  
Internet-Draft (work in progress) <draft-lhotka-yang-json-01>, CZ.NIC, June 2012.