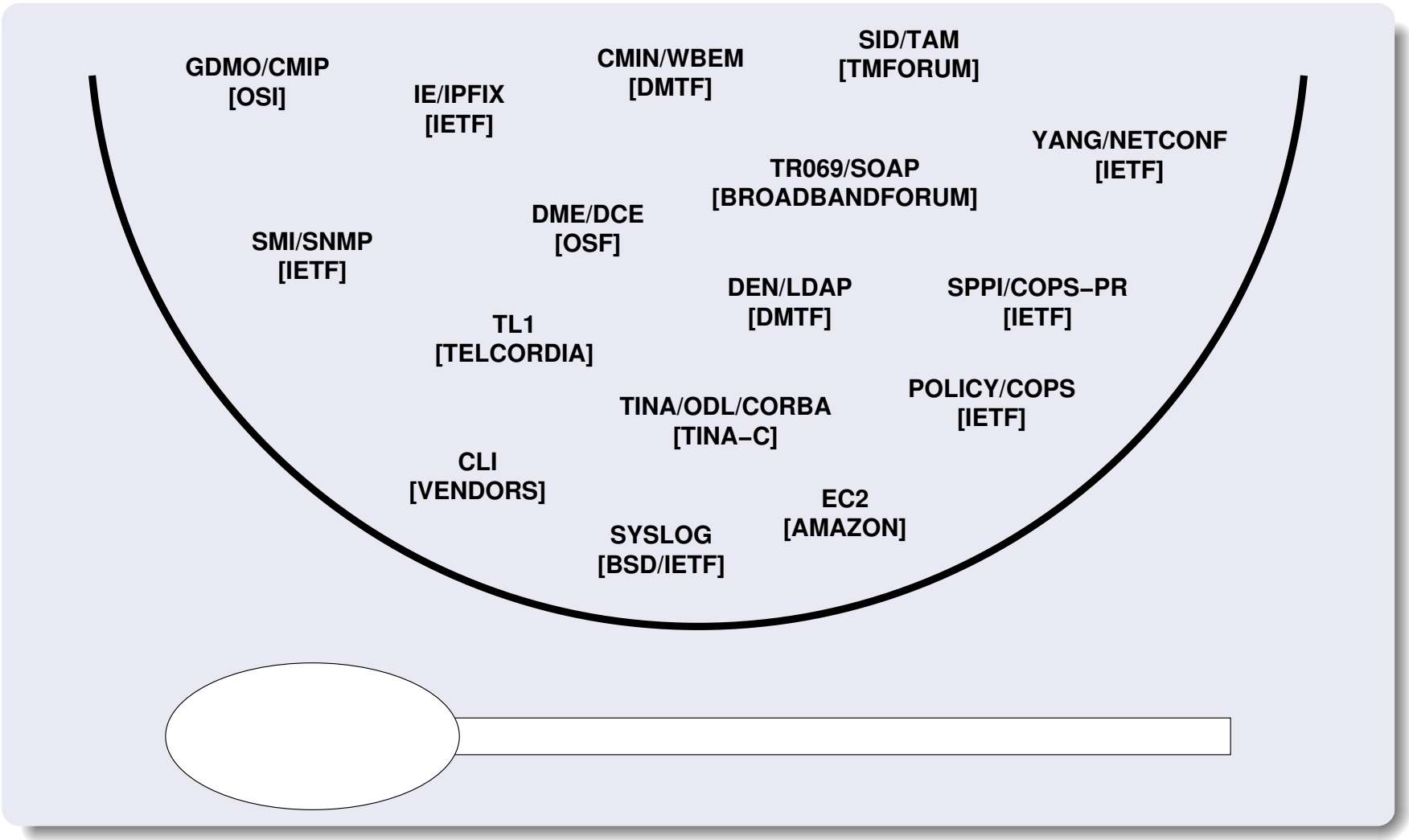
Network Configuration Management with NETCONF and YANG

Jürgen Schönwälder

84th IETF Meeting, Vancouver, 2012-07-29

JACOBS
UNIVERSITY

Network Management Protocol Soup



- See RFC 6632 for further details about the IETF's contribution to the network management protocol soup.

Today's Menu ...

NETCONF

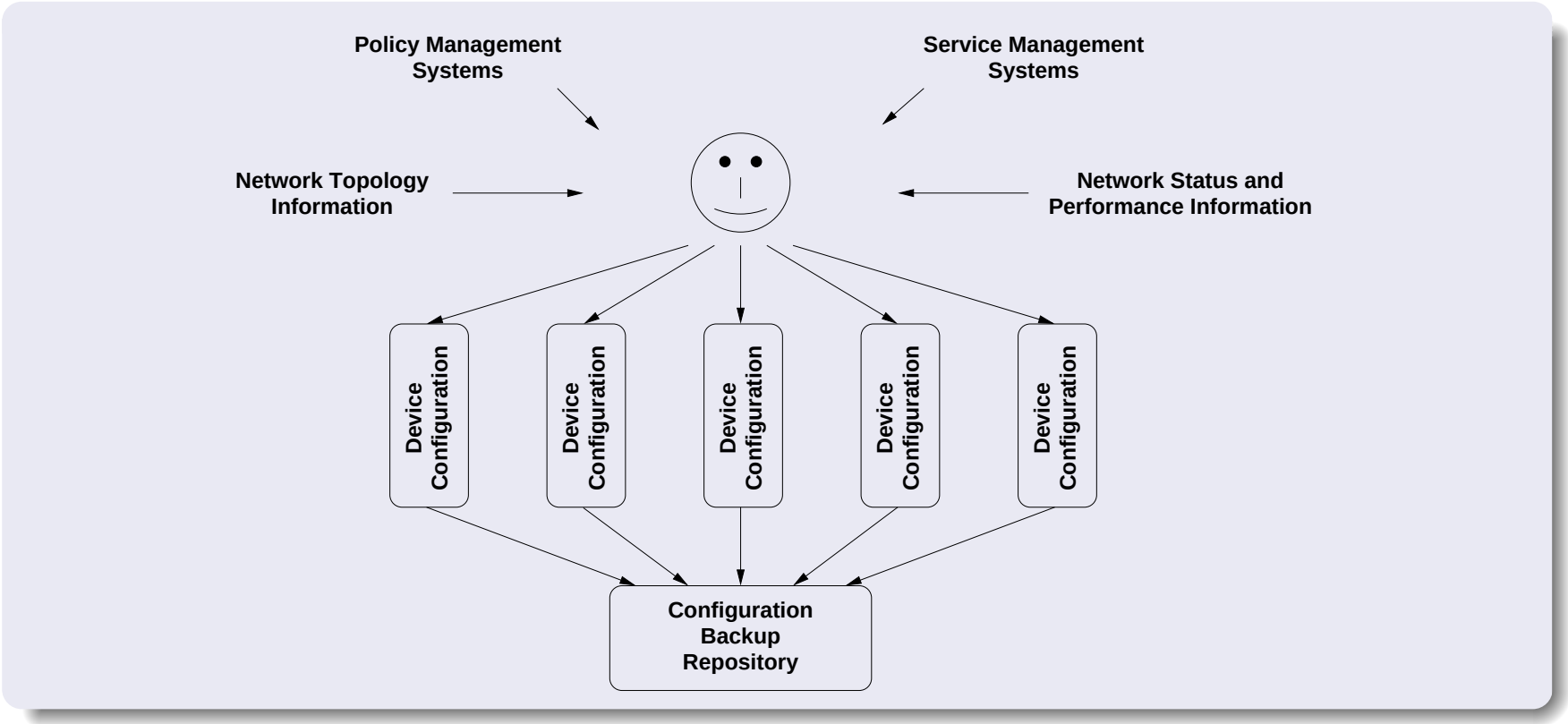
The Network Configuration Protocol (NETCONF) provides mechanisms to install, manipulate, and delete the configuration of network devices. [RFC 6241]

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]

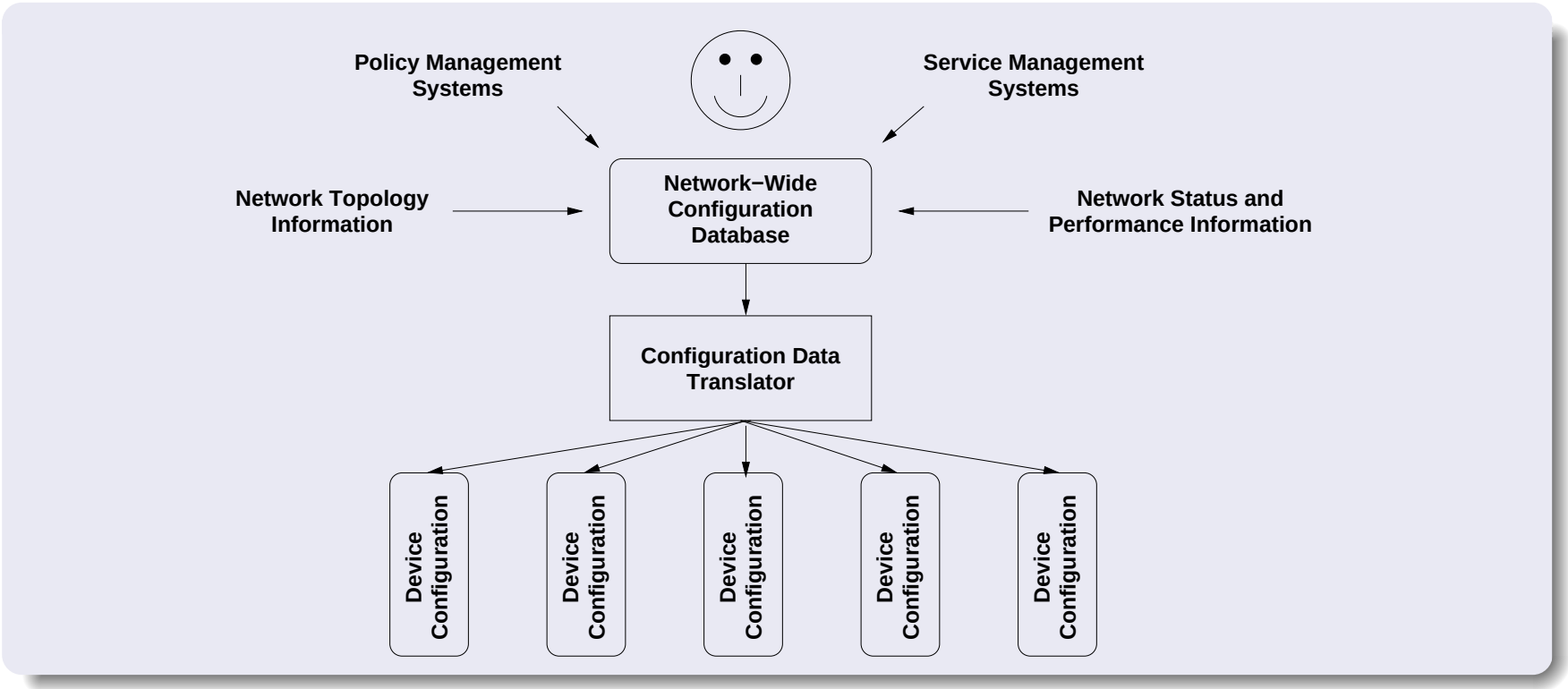
Configuration Management [15 min]

“The Network is the Record”



- ▷ 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

Configuration Management Requirements

R1 A configuration management protocol must be able to distinguish between configuration state and operational state. (configuration state vs. operational state)

- Configuration state:
  - Everything explicitely 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)

Configuration Management Requirements

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

Configuration Management Requirements

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

Configuration Management Requirements

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

Configuration Management Requirements

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

Configuration Management Requirements

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

Configuration Management Requirements

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

Configuration Management Requirements

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

Configuration Management Requirements

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

Configuration Management in the IETF

No	Description	SNMP	NETCONF
R1	config vs. oper state	-	+
R2	concurrency support	o	+
R3	config transactions	-	[+]
R4	multiple configs	-	[+]
R5	distribution vs. activation	-	[+]
R6	persistence of config state	o	+
R7	config change notifications	-	+
R8	config dump and restore	-	+
R9	support of standard tools	-	+

- These requirements originate from an IAB workshop that paved the way for NETCONF and YANG [RFC3535]

NETCONF and YANG Timeline

Date	Milestone
Jun 2002	IAB network management workshop
May 2003	NETCONF working group established
	— *work work work* —
Dec 2006	NETCONF 1.0 RFCs published
Apr 2008	NETMOD working group established
	— *work work work* —
Oct 2010	YANG 1.0 RFCs published
Jun 2011	NETCONF 1.1 RFCs published
	— *tutorial IETF 84* —
Q4 2012	Core configuration data models published

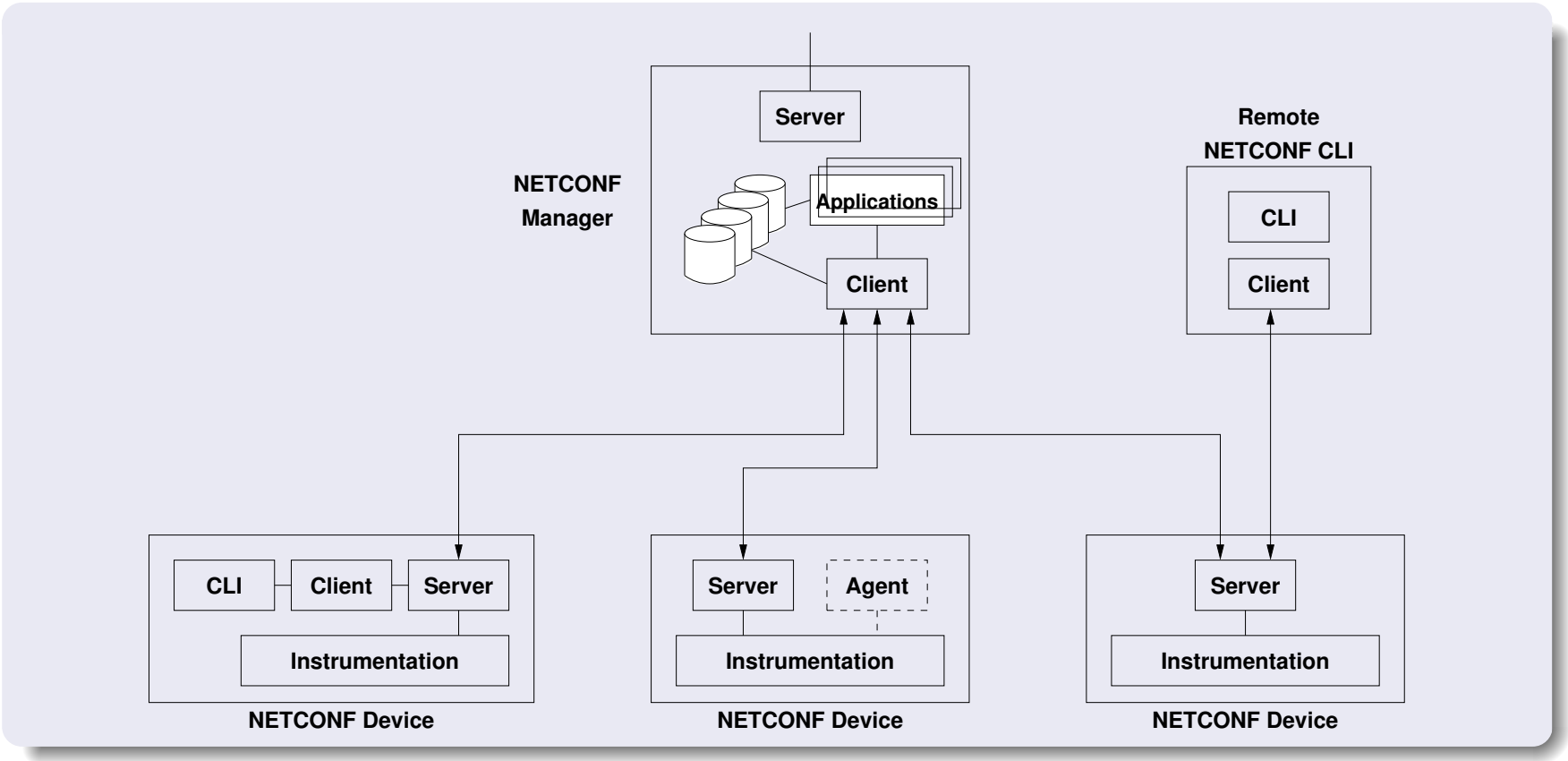
NETCONF Protocol [20 min]

NETCONF RFCs (as of today)

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

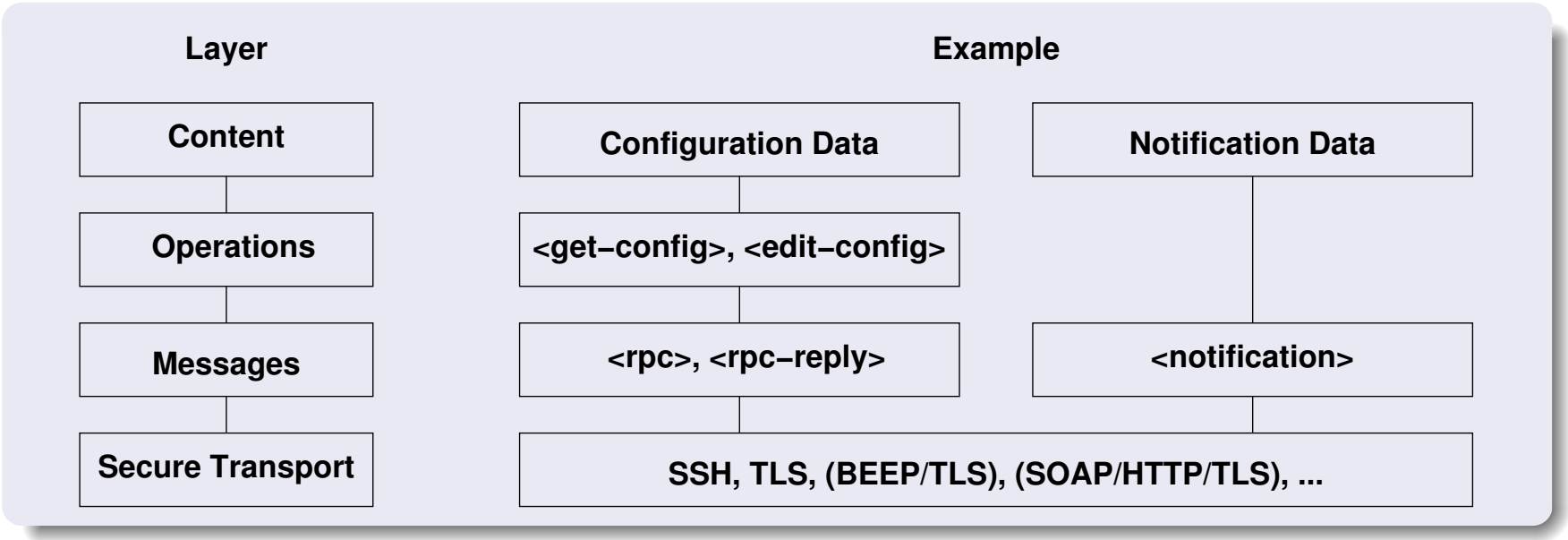
- Further RFCs define additional transports
- Some of the transports may be declared historic soon

Deployment Model



- NETCONF enabled devices include a NETCONF server
- Management applications include a NETCONF client
- Device CLIs can be a wrapped around a NETCONF client

Layering Model (RFC6241)



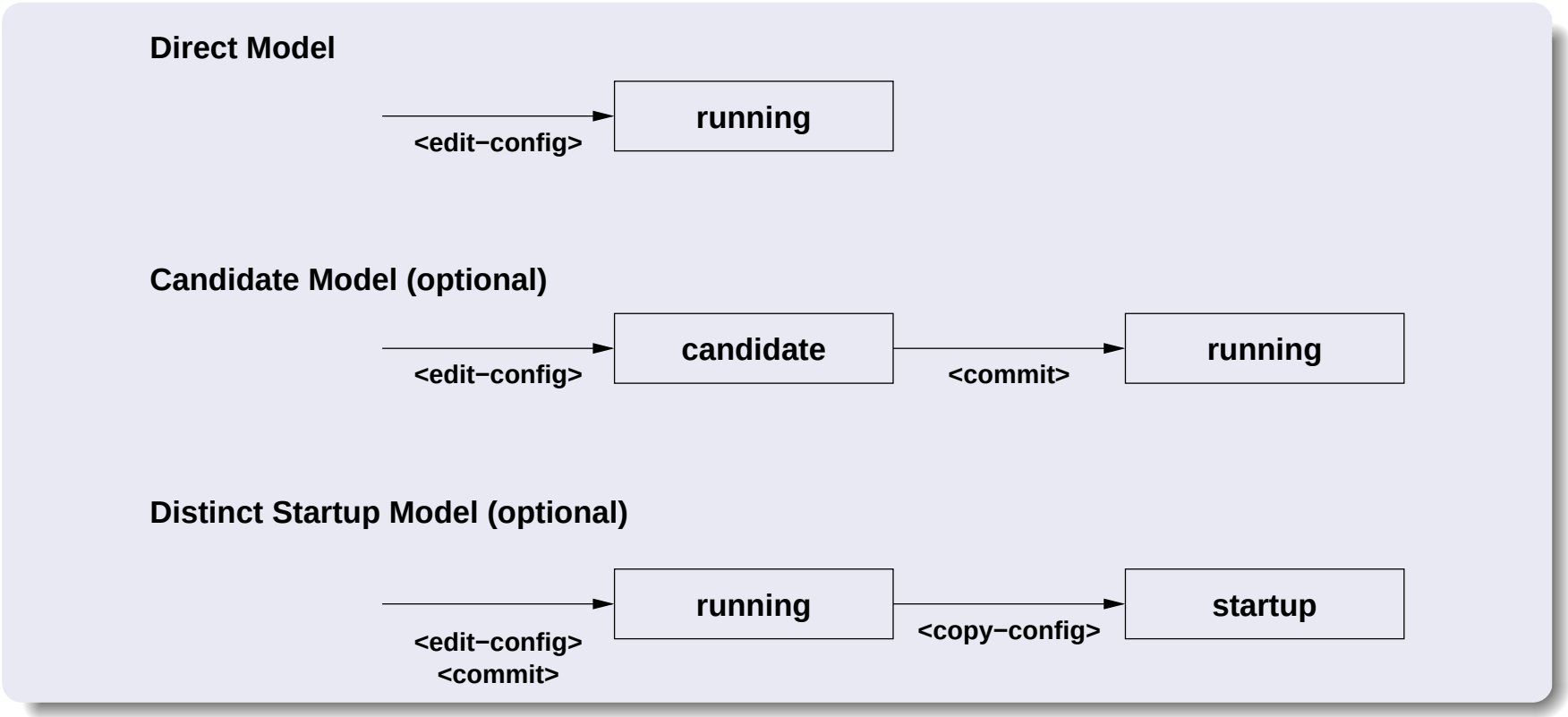
- Security is provided by the transport layer
- Operations focus on primitives to manage configurations
- The set of operations is designed to be extensible

Configuration Datastores (RFC6241)

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

Transaction Models (RFC6241)



- Some operations (`edit-config`) may support different selectable error behaviours, including rollback behaviour

Capability Exchange (RFC6241)

After estabilishing 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:       urn:ietf:params:xml:ns:netconf:base:1.1
S:     </capability>
S:     <capability>
S:       urn:ietf:params:xml:ns:netconf:capability:startup:1.0
S:     </capability>
S:     <capability>
S:       urn:ietf:params:xml:ns:yang:ietf-interfaces?
S:         module=ietf-interfaces&amp;revision=2012-04-29
S:     </capability>
S:   </capabilities>
S:   <session-id>4<session-id>
S: </hello>
```

Remote Procedure Calls (RFC6241)

The Remote Procedure Call (RPC) protocol consists of a <rpc/> message followed by an <rpc-reply/> message.

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

S: <rpc-reply message-id="101"
S:      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
S:   <data><!-- ...contents here... --></data>
S: </rpc-reply>
```

Remote Procedure Calls (RFC6241)

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)` $\rightarrow$ `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`.

NETCONF Operations (RFC6241)

- `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).

Editing (Patching) Configuration

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.

Editing (Patching) Configuration Example

```
C:  <rpc message-id="101"
C:       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
C:    <edit-config>
C:      <target>
C:        <running/>
C:      </target>
C:      <config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
C:        <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
C:          <interface nc:operation="replace">
C:            <name>Ethernet0/0</name>
C:            <mtu>1500</mtu>
C:          </interface>
C:        </interfaces>
C:      </config>
C:    </edit-config>
C:  </rpc>
```

Subtree Filtering

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.

Subtree Filtering Example

```
<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>
```

NETCONF over SSH (RFC6242)

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 initiall `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

NETCONF over SSH: End-of-Message Framing

```
S: <?xml version="1.0" encoding="UTF-8"?>
S: <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
S:   <capabilities>
S:     <capability>
S:       urn:ietf:params:xml:ns:netconf:base:1.1
S:     </capability>
S:     <capability>
S:       urn:ietf:params:xml:ns:netconf:capability:startup:1.0
S:     </capability>
S:     <capability>
S:       urn:ietf:params:xml:ns:yang:ietf-interfaces?
S:         module=ietf-interfaces&amp;revision=2012-04-29
S:     </capability>
S:   </capabilities>
S:   <session-id>4<session-id>
S: </hello>
S: ]]>]]>
```

- ▷ Server announces protocol version, startup capability, and the ietf-interfaces data model

NETCONF over SSH: End-of-Message Framing

```
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:     </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

NETCONF over SSH: Chunked Framing

```
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: #174
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: #174
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: #40
S:     </interfaces>
S:   </data>
S: </rpc-reply>
S: ##
```

YANG Data Modeling Language [20 min]

YANG RFCs (as of today)

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

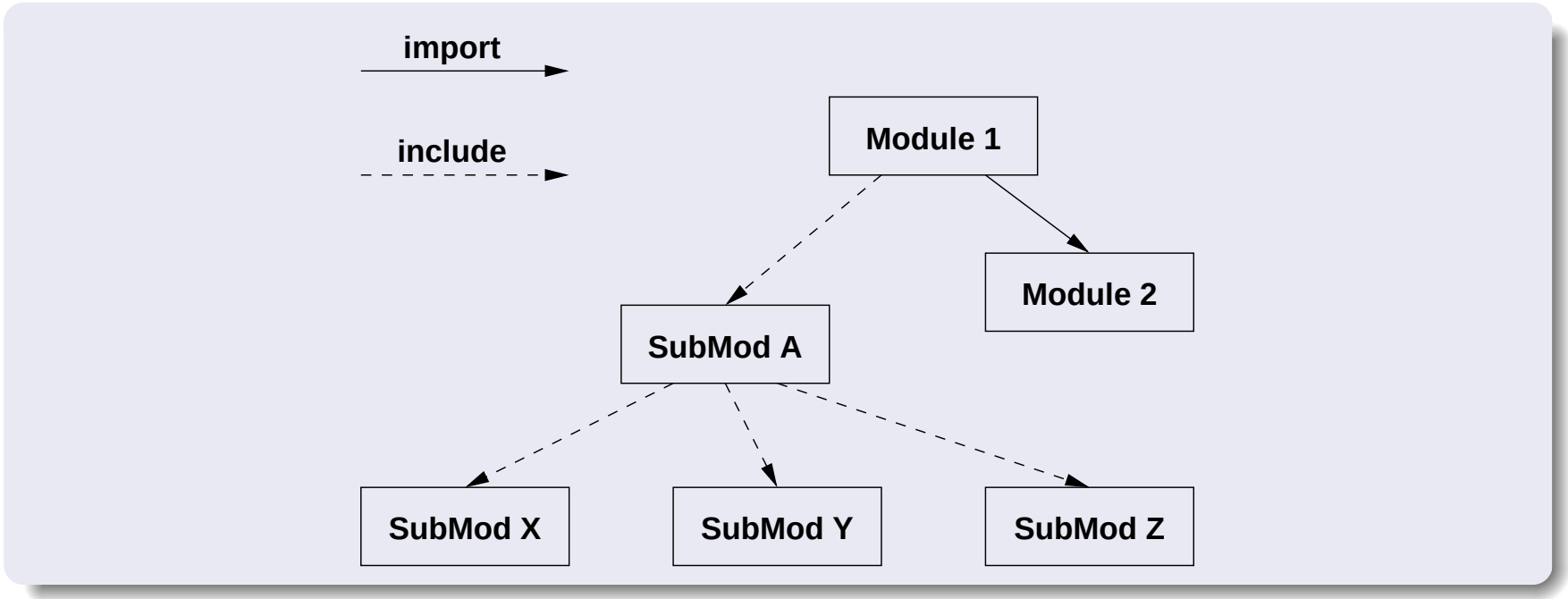
- YANG data models can be found in RFC 6022, RFC 6241, RFC 6243, RFC 6470, RFC 6536, ...
- We expect this list to grow soon with your help

YANG, YIN, XSD, RELAX NG, Schematron

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

Modules and submodules



- 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 Example

```
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";
  }
}
```

Built-in Data Types

Category	Types	Restrictions
Integral	{u,}int{8,16,32,64}	range
Decimals	decimal64	range, fraction-digits
String	string	length, pattern
Enumeration	enumeration	enum
Bool and Bits	boolean, bits	
Binary	binary	length
References	leafref	path
References	identityref	base
References	instance-identifier	
Other	empty	

- 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

Derived Types: typedef

```
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}]+)?';
        }
    }

    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.";
    }
}
```

Data Nodes: Leafs, Leaf-lists, Container, Lists

- `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

```
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>
```

Example: `list`

```
    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 existance 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

```
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

```
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, Choice, Notification, RPC

- `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;
            }
        }
    }
}
```

Example: notification

```
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>
```

Example: rpc

```
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>
```

Core Configuration Data Models [15 min]

Core Interfaces Data Model

```
+--rw interfaces
 +--rw interface [name]
    +--rw name                          string
    +--rw description?                  string
    +--rw type                          ianaift:iana-if-type
    +--rw location?                     string
    +--rw enabled?                      boolean
    +--ro if-index                      int32
    +--rw mtu?                          uint32
    +--rw link-up-down-trap-enable?     enumeration
```

- 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

Core IP Interfaces Data Model

```
+--rw if:interfaces
 +--rw if:interface [name]
    ...
    +--rw ipv4
       +--rw enabled?              boolean
       +--rw ip-forwarding?        boolean
       +--rw address [ip]
          +--rw ip                 inet:ipv4-address
          +--rw (subnet)?
             +--:(prefix-length)
             |  +--rw ip:prefix-length?   uint8
             +--:(netmask)
                +--rw ip:netmask?        inet:ipv4-address
```

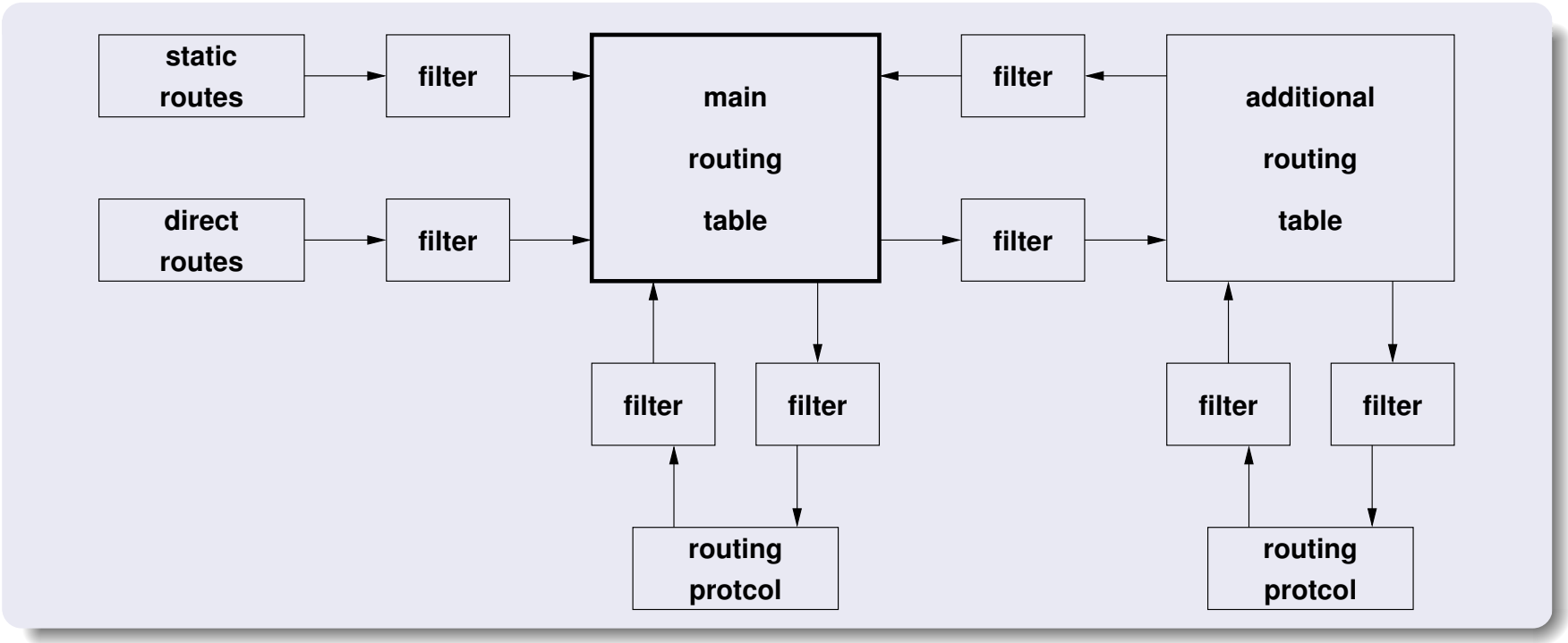
- IPv4 interface configuration augments interfaces
- The `(subnet)?` line indicates a `choice` and the `(prefix-length)` and `(netmask)` lines indicate a `case`
- Both, `choice` and `case` do not appear in the config

Core IP Interfaces Data Model

```
+--rw if:interfaces
 +--rw if:interface [name]
    ...
    +--rw ipv6
       +--rw enabled?                boolean
       +--rw ip-forwarding?          boolean
       +--rw address [ip]
       |  +--rw ip                   inet:ipv6-address
       |  +--rw prefix-length?       uint8
       +--rw dup-addr-detect-transmits?   uint32
       +--rw autoconf
          +--rw create-global-addresses?        boolean
          +--rw create-temporary-addressed?     boolean
          +--rw temporary-valid-lifetime?       uint32
          +--rw temporary-preferred-lifetime?   uint32
```

- IPv6 interface configuration similarily augments interfaces

Core IP Routing Data Model



- 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

Core IP Routing Data Model (ietf-routing)

```
+--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

Core System Data Model: System Time Mgmt

```
+--rw clock
|  +--ro current-datetime?      yang:date-and-time
|  +--ro boot-datetime?         yang:date-and-time
|  +--rw (timezone)?
|     +--:(timezone-location)
|     |  +--rw timezone-location?     iana:timezone
|     +--:(timezone-utc-offset)
|        +--rw timezone-utc-offset?   int16
+--rw ntp
|  +--rw use-ntp?      boolean
|  +--rw ntp-server [address]
|     +--rw address    inet:host
|     +--rw enabled?   boolean
```

- Information about current and boot time
- Timezone configuration
- Basic NTP client configuration

Core System Data Model: User Authentication

```
+--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

Description	Status
NETCONF Monitoring Data Model	RFC6022
NETCONF Access Control Data Model	RFC6536
IPFIX Configuration Data Model	RFC Editor
SNMP Configuration Data Model	WG Draft

- 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 [10 min]

NETCONF Implementations

Commercial (not necessarily complete)

Applied Informatics	http://www.appinf.com/
Centered Logic	http://www.centeredlogic.com/
MG-Soft	http://www.mg-soft.si/
Oracle	http://www.oracle.com/
Tail-f	http://www.tail-f.com/
WebNMS	http://www.webnms.com/
YumaPro	http://www.yumaworks.com/

Open Source (not necessarily complete)

EnSuite	http://ensuite.sourceforge.net/
ncclient	http://code.google.com/p/ncclient/
netconfx	http://www.centeredlogic.com/
netconf4j	https://github.com/dana-i2cat/netconf4j
netconf4android	http://code.google.com/p/netconf4android/
netopeer	http://code.google.com/p/netopeer/
Yuma	http://sourceforge.net/projects/yuma/

NETCONF in Devices

Device Vendors (not necessarily complete)

Alaxala	http://www.alaxala.com/
Telco Systems	http://www.telco.com/
BigBand/Arris	http://www.arrisi.com/
Brocade	http://www.brocade.com/
Cisco Systems	http://www.cisco.com/
Edgeware	http://www.edgeware.tv/
Ericsson	http://www.ericsson.com/
H3C	http://www.h3c.com/
Huawei	http://www.huawei.com/
Juniper Networks	http://www.juniper.net/
Nexor	http://www.nexor.com/
RuggedCom	http://www.ruggedcom.com/
Sonus	http://www.sonus.net/
Taseon	http://www.taseon.com/
Verivue	http://www.verivue.com/

YANG Implementations

Commercial (not necessarily complete)

MG-Soft	`http://www.mg-soft.si/`
SNMP Research	`http://www.snmp.com/`
Tail-f (ConfD)	`http://www.tail-f.com/`
YumaPro	`http://www.yumaworks.com/`

Open Source (not necessarily complete)

jyang	`http://jyang.gforge.inria.fr/`
libsmi	`http://www.ibr.cs.tu-bs.de/projects/libsmi/`
pyang	`http://code.google.com/p/pyang`
Yuma	`http://sourceforge.net/projects/yuma/`

YANG Data Models

IETF (not necessarily complete)

IPFIX	IPFIX Configuration Model
NETMOD	Interfaces Core Configuration Model
NETMOD	IP and Routing Core Data Models
NETMOD	System Configuration Data Model
NETMOD	SNMP Configuration Data Model
NETCONF	NETCONF Monitoring Data Model
NETCONF	NETCONF Access Control Data Model

Other SDOs (not necessarily complete)

Open Networking Found.	OF-Config 1.1
CableLabs	CM-SP-CCAP-OSSI-I02-120329
Metro Ethernet Forum	PM and FM (to be published)

Future Directions [10 min]

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

```
{
    "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
            }
        ]
    }
}
```

YANG to JSON Example (XML)

```
<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)

RESTful API for YANG

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 to allow batching of edits and handling of concurrent editing transactions.

RESTful API for YANG Example

```
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:         "urn:ietf:params:xml:ns:yang:ietf-yang-api?module=ietf-yang-api&
S:       ]
S:     },
S:     "version": "1.0"
S:   }
S: }
```

RESTful API for YANG Example

```
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 [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

Reading Material I

[1] J. Schönwälder, M. Björklund, and P. Shafer.
Network Configuration Management Using NETCONF and YANG.
IEEE Communications Magazine, 48(9):166–173, September 2010.

[2] P. Shafer.
An Architecture for Network Management Using NETCONF and YANG.
RFC 6244, Juniper Networks, June 2011.

[3] J. Schönwälder.
Overview of the 2002 IAB Network Management Workshop.
RFC 3535, International University Bremen, May 2003.

[4] L. Sanchez, K. McCloghrie, and J. Saperia.
Requirements for Configuration Management of IP-based Networks.
RFC 3139, Megisto, Cisco, JDS Consultant, June 2001.

[5] M. Ersue and B. Claise.
An Overview of the IETF Network Management Standards.
RFC 6632, Nokia Siemens Networks, Cisco Systems, June 2012.

[6] M. Bjorklund.
YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF).
RFC 6020, Tail-f Systems, October 2010.

[7] J. Schönwälder.
Common YANG Data Types.
RFC 6021, Jacobs University, October 2010.

[8] A. Bierman.
Guidelines for Authors and Reviewers of YANG Data Model Documents.
RFC 6087, Brocade, January 2011.

Reading Material II

[9] L. Lhotka.
Mapping YANG to Document Schema Definition Languages and Validating NETCONF Content.
RFC 6110, CESNET, February 2011.

[10] J. Schönwälder.
Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules.
RFC 6643, Jacobs University, July 2012.

[11] B. Linowski, M. Ersue, and S. Kuryla.
Extending YANG with Language Abstractions.
RFC 6095, Nokia Siemens Networks, 360 Treasury Systems, March 2011.

[12] R. Enns, M. Bjorklund, J. Schönwälder, and A. Bierman.
Network Configuration Protocol (NETCONF).
RFC 6241, Juniper Networks, Tail-f Systems, Jacobs University, Brocade, June 2011.

[13] M. Wasserman.
Using the NETCONF Protocol over Secure Shell (SSH).
RFC 6242, Painless Security, June 2011.

[14] A. Bierman and B. Lengyel.
With-defaults Capability for NETCONF.
RFC 6243, Brocade, Ericsson, June 2011.

[15] A. Bierman.
Network Configuration Protocol (NETCONF) Base Notifications.
RFC 6470, Brocade, February 2012.

[16] A. Bierman and M. Bjorklund.
Network Configuration Protocol (NETCONF) Access Control Model.
RFC 6536, YumaWorks, Tail-f Systems, March 2012.

Reading Material III

[17] S. Chisholm and H. Trevino.
NETCONF Event Notifications.
RFC 5277, Nortel, Cisco, July 2008.

[18] M. Scott and M. Bjorklund.
YANG Module for NETCONF Monitoring.
RFC 6022, Ericsson, Tail-f Systems, October 2010.

[19] B. Lengyel and M. Bjorklund.
Partial Lock Remote Procedure Call (RPC) for NETCONF.
RFC 5717, Ericsson, Tail-f Systems, December 2009.

[20] M. Bjorklund.
A YANG Data Model for Interface Configuration.
Internet-Draft (work in progress) <draft-ietf-netmod-interfaces-cfg-05>, Tail-f Systems, July 2012.

[21] M. Bjorklund.
A YANG Data Model for IP Configuration.
Internet-Draft (work in progress) <draft-ietf-netmod-ip-cfg-05>, Tail-f Systems, July 2012.

[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.
Internet-Draft (work in progress) <draft-ietf-netmod-system-mgmt-02>, YumaWorks, Tail-f Systems, July 2012.

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

Reading Material IV

[25] A. Bierman and M. Bjorklund.
YANG-API Protocol.
Internet-Draft (work in progress) <draft-bierman-netconf-yang-api-00>, YumaWorks, Tail-f Systems, May 2012.