Network Configuration Management Using NETCONF and YANG

Jürgen Schönwälder, Jacobs University
Martin Björklund, Tail-f Systems
Phil Shafer, Juniper Networks

ABSTRACT

The Internet Engineering Task Force has standardized a new network configuration management protocol called NETCONF, which provides mechanisms to install, manipulate, and delete the configuration of network devices. This article describes the NETCONF protocol and a recently introduced NETCONF data modeling language called YANG. The YANG language allows data modelers to define the syntax and semantics of device configurations, and supports translations to several XML schema languages.

INTRODUCTION

The management of the configuration of a large number of networked devices remains a highly important practical problem. Device configurations and the mechanisms to retrieve and modify them are largely vendor-specific, and the most widely used configuration interfaces today are proprietary command line interfaces (CLIs), making it costly to achieve a high level of efficiency and reliability through automation. In 2003 the Internet Engineering Task Force (IETF) started an effort to develop and standardize a network configuration management protocol, which led to the publication of the Network Configuration (NETCONF) protocol [1] at the end of 2006.

The NETCONF protocol supports several features required for configuration management that were lacking in other network management protocols such as Simple Network Management Protocol (SNMP) [2]. NETCONF operates on so-called datastores and represents the configuration of a device as a structured document, serialized using the Extended Markup Language (XML). The protocol distinguishes between running configurations, startup configurations, and candidate configurations. In addition, it provides primitives to assist with the coordination of concurrent configuration change requests and support distributed configuration change transactions over several devices. Finally, NETCONF provides filtering mechanisms, validation capabilities, and event notification support.

The work in the IETF initially focused on the protocol design and its specification. It was, however, clear that a common data modeling language is needed in addition to the protocol to express the structure and semantics of configuration information in a vendor-neutral format. A proposal for a NETCONF data modeling language called YANG was developed in 2007 and is being standardized in the IETF since 2008.

The aim of this article is threefold. First, we provide an overview of the NETCONF protocol. Second, we describe the recently defined YANG data modeling language. Finally, we discuss some of the available implementations and how they have been used to provide a programmatic configuration interface that integrates well with other management interfaces of devices.

The rest of the article is structured as follows. The next section provides additional background information before the NETCONF protocol is described. The YANG data modeling language is then introduced. Implementation experience is reported and related work is discussed before the article concludes in the final section.

BACKGROUND AND MOTIVATION

In 2002 the Internet Architecture Board (IAB) organized a workshop in order to guide future network management standardization activities in the IETF. The workshop was attended by network operators and protocol developers, and resulted in several concrete recommendations [3]. One of the recommendations was to focus IETF resources on the development of standards for network device configuration management. Another recommendation was to use XML for data encoding purposes.

In 2003 a working group was formed in the Operations and Management area of the IETF to produce a protocol supporting network configuration. The working group charter mandated that XML be used for data encoding purposes. The protocol produced by this working group is called NETCONF [1]. The design of NETCONF has been influenced by proprietary protocols such as Juniper Networks' JUNOScript application programming interface (API).

Network operators today use differing approaches to managing device configurations

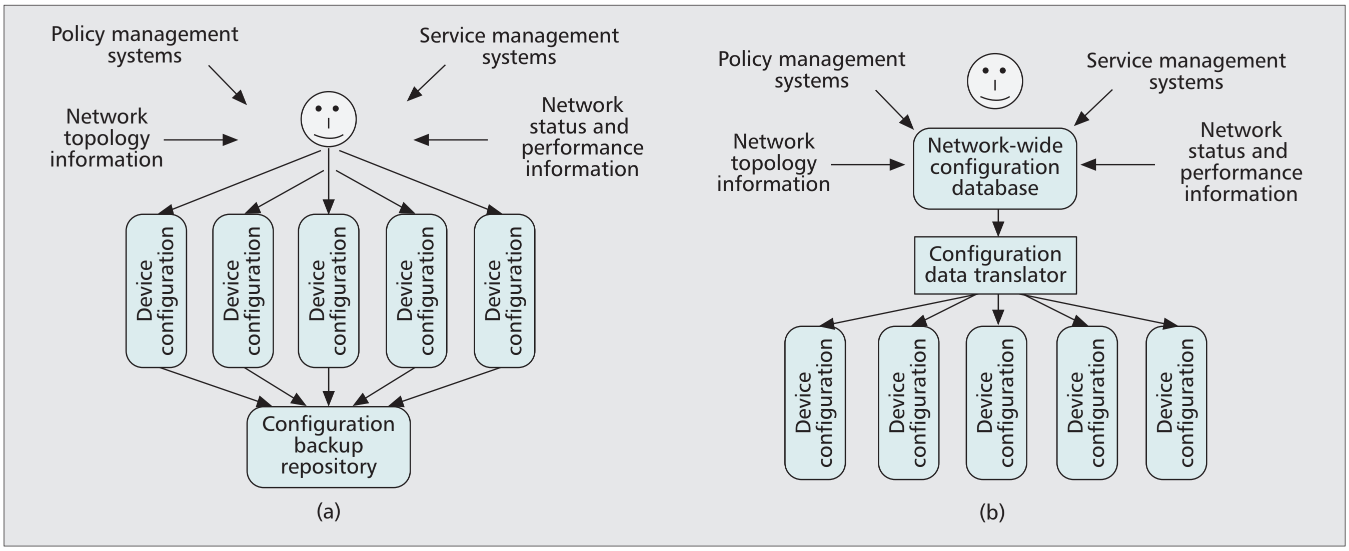
Policy management systems
Service management systems
Network topology information
Network status and performance information
Device configuration
Device configuration
Device configuration
Device configuration
Device configuration
Configuration backup repository
(a)
Policy management systems
Service management systems
Network topology information
Network-wide configuration database
Network status and performance information
Configuration data translator
Device configuration
Device configuration
Device configuration
Device configuration
Device configuration
(b)

Figure 1. *The Network is the Record vs. the Generate Everything approaches to configuration management: a) Network is the Record; b) Generate Everything.*

(Fig. 1). In the *Network is the Record* approach, operators directly modify the configuration of devices. To make the configuration change process (and any errors incurred due to configuration changes) trackable, all configurations are copied from the devices into a configuration backup repository. In the *Generate Everything* approach, a network-wide configuration database is used to generate device configurations that are pushed to the devices. In both approaches it is necessary to be able to push configuration changes to devices and dump/restore complete configurations. In addition, it is vital that devices distinguish between configuration data and data describing operational state that has been obtained via other means (e.g., via routing protocols or signaling protocols).

The driving force behind NETCONF is the need for a programmatic cross-vendor interoperable interface to manipulate configuration state. The approach of automating CLIs using programs and scripts has proven problematic, especially when it comes to maintenance and versioning issues. Several operators reported during the IAB workshop that they find it time consuming to maintain programs or scripts that interface with different versions of a CLI.

Figure 2 shows a NETCONF deployment scenario. It assumes that a network-wide configuration or policy system uses the NETCONF protocol to push configuration changes to NETCONF enabled devices. In such a deployment a policy-driven network manager acting as a policy decision point includes a NETCONF client. The managed devices include a NETCONF server acting as a policy enforcement point. Of course, the setup shown in Fig. 2 requires that a policy manager can translate higher-level policies into device configurations. How such a translation is done is not considered part of IETF standardization activities; NETCONF only provides the protocol to communicate either complete configurations or configuration changes to devices in a robust and scalable manner.

The right part of Fig. 2 shows a CLI that talks NETCONF to a server in order to implement the functionality provided through the CLI. NETCONF is designed to be powerful enough to drive CLIs. Cost savings on the device vendor side can only be achieved if there is a single method to effect configuration changes, which can be shared across programmatic and human operator interfaces. This implies that the scope of the NETCONF protocol is actually broader than just device configuration.

Network Configuration Protocol

The NETCONF protocol [1] has a simple layered architecture shown in Fig. 3. The core of NETCONF is a simple remote procedure call (RPC) layer running over secure transports such as SSH, TLS, SOAP, or BEEP. Secure Shell (SSH) [4] transport is mandatory to implement as a means of promoting interoperability. The operations layer residing on top of the RPC layer provides specific operations to manipulate configuration state. The configuration data itself forms the content layer residing above the operations layer. The NETCONF specification mainly deals with generic operations to retrieve and modify configuration state. An additional document [5] defines operations to subscribe to notification streams and receive notifications. Further operations are expected to be added in the future in order to support data-model-specific management operations.

NETCONF assumes that the configuration state of a device can be represented as a structured document that can be retrieved and manipulated (document-oriented approach). In order to deal with large configurations, the protocol supports filtering mechanisms that allow clients to retrieve only a subset of the configuration.

NETCONF supports multiple configuration datastores. A configuration datastore contains all information needed to get a device from its initial default state into the desired configura-

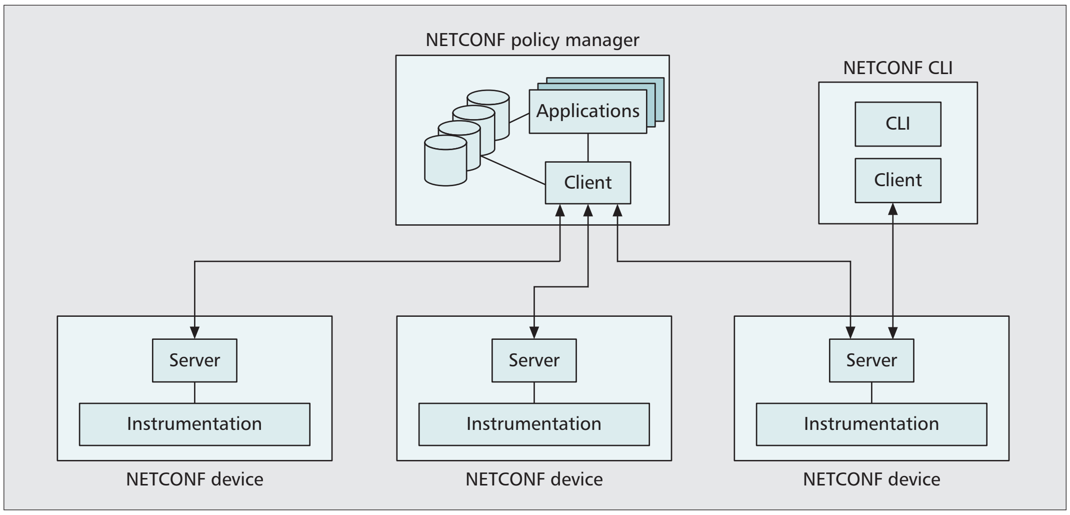

Figure 2. *NETCONF deployment scenario including a policy manager and a CLI.*

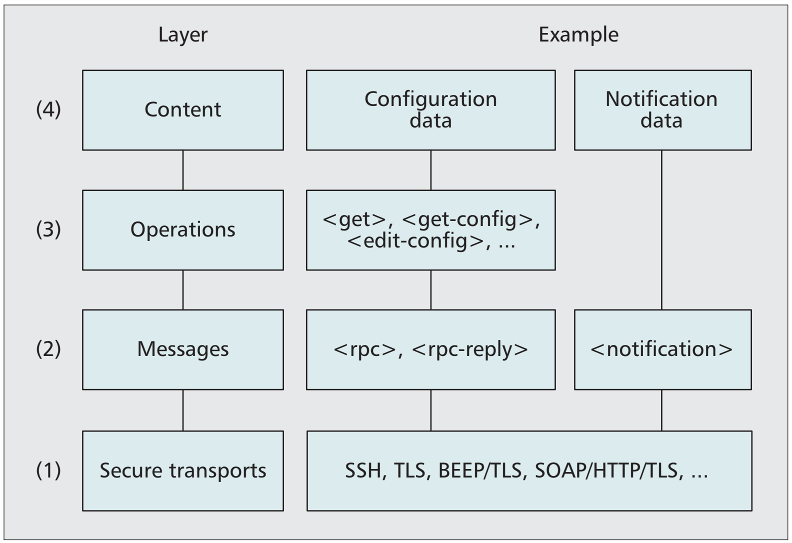

Figure 3. *NETCONF protocol layers.*

tion state. The `running` datastore is always present and describes the currently active configuration. In addition, NETCONF supports the notion of a `startup` configuration datastore, which is loaded by the device as part of its initialization when it reboots or reloads, and a `candidate` datastore, which is a scratch buffer that can be manipulated and later committed to the `running` datastore.

NETCONF features a rich set of protocol operations. It is generally expected that new protocol operations will be added in the future by vendors and standardization bodies. This essentially means that NETCONF can easily support a command-oriented approach in addition to the already defined document-oriented approach to manipulating configuration state.

Table 1 shows the protocol operations that have been defined so far by the NETCONF working group of the IETF. The first six operations all operate on configurations stored in configuration datastores selected by the `source` or `target` arguments. The `lock` and `unlock` operations do coarse-grained locking, and locks are intended to be short-lived. More fine-grained locking mechanisms are currently being defined in the IETF.

The `get` operation is provided to retrieve a device's configuration state together with its operational state, while the `get-config` operation only returns configuration state. The distinction between operational state and configuration state is a very important feature of NETCONF missing from other network management protocols, such as SNMP, where it is assumed that management applications have the necessary knowledge to identify which data item belongs to the operational or configuration state.

The `get` and `get-config` operations both support an optional `filter` parameter to select the subset of the configuration and state data that should be retrieved. Implementations can support different filter mechanisms. The mandatory subtree filter mechanism selects the branches of an XML tree matching a provided template. As an optional feature, implementations can choose to support XPATH [6] expressions as filters.

The most powerful and also most complex operation is the `edit-config` operation. The `edit-config` operation can be used to modify the content of a configuration datastore by creating, deleting, replacing, or merging configuration elements. In a nutshell, `edit-config` works by applying a patch to a datastore in order to generate a new configuration. Since a tree-based representation is used to represent configuration state, it is necessary to describe which branches in the tree should be created, deleted, replaced, or merged. NETCONF solves this by adding an `operation` attribute to the XML payload of the `edit-config` operation. With this approach, relatively complex configuration changes can be achieved in a single `edit-config` invocation.

Notification support in NETCONF is based on an event stream abstraction. The event stream abstraction enables NETCONF to support several different event sources. Clients interested in receiving notifications subscribe to event streams. On systems that maintain event logs, it is possible to subscribe to an event stream at some time in the past and the device will playback all recorded notifications at the beginning of the notification stream. A subscription to an event stream establishes a filter that is applied before event notifications are sent to the client. This allows one to select only the relevant notifications and improves scalability.

Finally, there are some housekeeping operations. The `close-session` operation initiates a graceful close of the current session, while the `kill-session` operation forces the termination of another session identified by the `session-id`.

NETCONF supports several different configuration change transaction models as shown in Fig. 4. The simplest model only requires a `running` configuration datastore, and all configuration changes are directly applied to the `running` configuration. The optional candidate model introduces a `candidate` configuration datastore acting as a scratchpad. Configuration changes on the `candidate` configuration have no effect until they are committed to the `running` configuration. To support configuration change transactions involving several devices, where the configuration change may lead to transient connectivity problems, a confirmed commit operation with automatic rollback is supported. Finally, the optional distinct startup model assumes the existence of a special `startup` configuration datastore that is loaded by a device when it reboots or reloads. By using the `copy-config` operation, the currently running configuration can be made the startup configuration.

The design of the NETCONF protocol is modular, and implementations with different capabilities are possible. To promote interoperability, a mechanism to exchange the capabilities or servers and clients, and announce the supported data models and any deviations is used. When a NETCONF session is established, both client and server send a `hello` message to announce the supported capabilities. Figure 5 shows a capability exchange followed by an `edit-config` operation.

Operation	Arguments
get-config	source [filter]
edit-config	target [default-operation] [test-option] [error-option] config
copy-config	target source
delete-config	target
lock	target
unlock	target
get	[filter]
close-session	
kill-session	session-id
discard-changes	
validate	source
commit	[confirmed confirm-timeout]
create-subscription	[stream] [filter] [start] [stop]

Table 1. *NETCONF protocol operations (arguments in brackets are optional).*

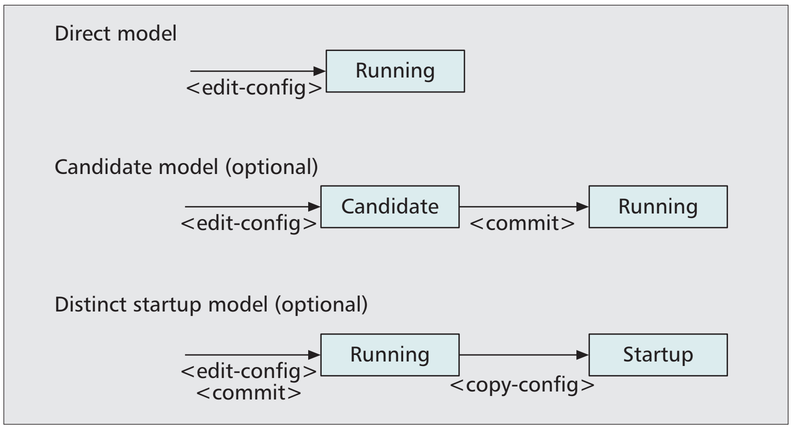

Figure 4. *NETCONF configuration change transaction models.*

Data Modeling Language YANG

Since NETCONF uses XML to encode network management data, it may seem obvious to use one of the existing XML schema languages to formally specify the format of these XML documents. While some parts of the industry favor the XML Schema Definition Language (XSD) [6], there is significant uptake of RelaxNG [7] in recent years. But putting aside the differences between XSD and RelaxNG, it is clear that additional NETCONF-specific information needs to be specified that goes well beyond the capabilities of these XML schema languages. Both XSD and RelaxNG only address part of the problem to be solved.

During the development of the NETCONF protocol specifications, which are formally defined using XSD, it has been observed that XSD notation is difficult to read and verify by humans. Other schema notations such as RelaxNG (and especially its compact notation) seem to be easier to read and write. Still, both schema languages tend to be relatively far away from an implementer's view of a configuration datastore and tend to become cumbersome to use when all the necessary NETCONF-specific extensions are added. Furthermore, the validation capabilities of standard tools have limited value; what is most urgently needed is the validation of configuration datastores and not so much the validation of individual protocol messages that contain the serialization of (parts of) a configuration datastore. Given the nature of the `edit-config` operation, individual messages might not satisfy all data model constraints but can still lead to a valid configuration datastore at commit time.

The YANG data modeling language [8] therefore takes a different approach. YANG aims to be a highly readable and compact domain-specif-

```
S: <?xml version="1.0" encoding="UTF-8"?>
S: <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
S:   <capabilities>
S:     <capability>urn:ietf:params:netconf:base:1.0</capability>
S:     <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
S:     <capability>urn:ietf:params:xml:ns:yang:ietf-inet-types?revision=2009-05-13</capability>
S:     <capability>http://acme.example.com/yang/acme-dns-resolver?revision=2009-08-12</capability>
S:   </capabilities>
S:   <session-id>232</session-id>
S: </hello>
C: <?xml version="1.0" encoding="UTF-8"?>
C: <nc:hello xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
C:   <nc:capabilities>
C:     <nc:capability>urn:ietf:params:netconf:base:1.0</nc:capability>
C:   </nc:capabilities>
C: </nc:hello>

C: <?xml version="1.0" encoding="UTF-8"?>
C: <nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1234">
C:   <nc:edit-config>
C:     <nc:target><nc:candidate/></nc:target>
C:     <nc:config>
C:       <dns xmlns="http://acme.example.com/yang/acme-dns-resolver">
C:         <resolver>
C:           <nameserver nc:operation="delete">
C:             <address>192.0.2.4</address>
C:           </nameserver>
C:           <nameserver nc:operation="create">
C:             <address>192.0.2.8</address>
C:             <port>5353</port>
C:           </nameserver>
C:         </resolver>
C:       </dns>
C:   </nc:config>
C:   </nc:edit-config>
C: </nc:rpc>
S: <?xml version="1.0" encoding="UTF-8"?>
S: <rpc-reply message-id="1234" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
S:   <ok/>
S: </rpc-reply>
```

Figure 5. *NETCONF capability exchange followed by an edit-config exchange; the prefix* S: *indicates the server and the prefix* C: *the client.*

ic language for defining NETCONF data models. YANG comes with a twin called YIN, which is an XML representation of YANG so that standard XML tools can be used to process YANG data model definitions. A lossless two-way conversion between YANG and YIN is defined. In addition, one-way conversions to XSD and RelaxNG are available so that corresponding tools can be used.

Modules and Types

All YANG definitions are contained in modules. A module is identified by its name and has its own XML namespace. A module can be further subdivided into submodules to simplify the maintenance of complex modules. The submodule structure, however, is not visible outside of the module; all submodules share the same XML namespace, and all definitions of all submodules are accessible by importing the module.

YANG features a small set of built-in data types and provides a library of commonly used derived data types. The data type derivation mechanisms of YANG are compatible with XSD to achieve simple translations between YANG and XSD or RelaxNG.

Figure 6 shows the `acme-dns-resolver` YANG module modeling a Domain Name System (DNS) resolver. The module imports definitions from the module `ietf-inet-types` and defines, among other things, the derived enumerated type `server-status`. Figure 7 shows possible configuration content consistent with the YANG data model.

Leafs, Container, Lists

YANG stores all data in the `leaf` elements of an XML tree. The leaf statement defines a simple leaf carrying a single value. A list of simple `leafs`, each carrying a single value, can be defined using the `leaf-list` statement. Intermediate nodes of the XML tree are defined using the `container` statement. The `list` statement can be used to define complex structures that can have multiple instances. A list may contain other `leafs`, `leaf-lists`, `containers`, or lists.

The resolver YANG module shown in Fig. 6 defines a `leaf domain` and a `leaf-list search` of domain suffixes to search through. Since the order of this list is relevant, it is marked to be ordered by the user. The list nameserver lists the DNS server addresses. The domain-name, search, and nameserver configuration elements are embedded in the container resolver. The resolver container is marked to contain configuration data and itself embedded in the dns container.

GROUPINGS, AUGMENTATIONS, CONSTRAINTS

The definition of the list nameserver makes use of the grouping server-address. A grouping can be seen as a reusable constructed type. In fact, a grouping can contain an arbitrary hierarchy of containers, lists, leafs, and leaf-lists. However, unlike constructed types in other languages, a grouping can be modified when it is used.

With the goal of standardized configuration interfaces in mind, it is necessary to provide language mechanisms allowing vendors to extend standardized definitions with vendor-specific definitions. The augment statement allows one to add definitions to some other part of the configuration tree, which might be defined by some external module. The example in Fig. 6 shows an augmentation adding a leaf status to the list nameserver in the container resolver.

The YANG language allows data modelers to formally express constraints that must be validated by NETCONF servers. There are two types of constraints: must constraints are used to express constraints (in the form of XPATH expressions) that must be satisfied by a valid configuration; when constraints are used to define sparse augmentations where nodes are only added when a condition (in the form of an XPATH expression) is true. Note that the usage of XPATH at module design time does not require that module implementations have to support full XPATH at runtime.

NOTIFICATIONS AND OPERATIONS

The YANG language supports the formal definition of notifications and operations. The notification statement defines a notification name and the content of the notification.

The rpc statement defines operations by giving them a name and defining the input and output parameters. The usual YANG data definition statements are used to define input and output parameters as well as notification content. By using the rpc statement, a YANG modeler can define a command-oriented interface that matches the features provided by CLIs.

FEATURES AND DEVIATIONS

The YANG language supports a mechanism to mark a portion of a data model as optional. This feature mechanism allows a data model designer to break the data model for a complex protocol into a set of optional features. YANG identifies features by name. The names of the features supported by a NETCONF server are advertised through the capability exchange mechanism when a NETCONF session is established.

In an ideal world all devices would be required to implement a data model exactly as defined, and deviations from the model would not be allowed. But in the real world, devices are often not able or willing to implement the model as written. YANG provides a mechanism

```
module acme-dns-resolver {

    namespace "http://acme.example.com/yang/acme-dns-resolver/1.0";
    prefix "acme-res";

    import "ietf-inet-types" { prefix "inet"; }

    organization "ACME Inc.";
    contact      "support@acme.example.com";
    description
       "The YANG module for configuring the name resolver
       library used by ACME products";

    revision "2009-08-12" {
       description "Initial revision.";
    }

    feature "status" {
       description
          "The status feature indicates that the server
          provides status information for the configured
          nameservers.";
    }

    typedef server-status {
       type enumeration {
          enum unknown;
          enum answering;
          enum failed;
       }
       description
          "This type represents the status of a server.";
    }

    grouping server-address {
       leaf address {
          type inet:ip-address;
       }
       leaf port {
          type inet:port-number;
       }
    }

    container dns {
       container resolver {
          config true;

          description
             "The configuration of the resolver library.";

          leaf domain {
             type inet:domain-name;
             description
                "The host name of this system.";
          }

          leaf-list search {
             type inet:domain-name;
             ordered-by user;
             description
                "List of domain names to search.";
          }

          list nameserver {
             key address;
             description
                "The list of known name servers.";

             uses server-address {
                refine port { default 53; }
             }
          }
       }
    }

    augment "/dns/resolver/nameserver" {
       leaf status {
          type server-status;
          config false;
          if-feature "status";
       }
    }
}
```

Figure 6. *YANG module to configure and monitor a DNS resolver.*

```
<?xml version="1.0" encoding="UTF-8"?>
<dns xmlns="http://acme.example.com/yang/acme-dns-resolver/1.0">
  <resolver>
    <domain>test.example.com</domain>
    <search>test.example.com</search>
    <search>example.com</search>
    <nameserver>
      <address>192.0.2.4</address>
      <status>answering</status>
    </nameserver>
  </resolver>
</dns>
```

Figure 7. *Configuration consistent with the DNS resolver data model.*

```
#! /usr/bin/env python2.6

import sys, os, warnings
from ncclient import manager

template = """<config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <dns xmlns="http://acme.example.com/yang/acme-dns-resolver/1.0">
  <resolver><nameserver nc:operation="%s">
  <address>%s</address>
  </nameserver></resolver></dns></config>"""

def demo(host, user, changes):
    with manager.connect(host=host, port=22, username=user) as m:
        assert(":candidate" in m.server_capabilities)
        with m.locked(target='candidate'):
            m.discard_changes()
            for c in changes:
                m.edit_config(target='candidate', config=template % c)
                m.commit()

if __name__ == '__main__':
    for device in sys.argv[1:]:
        demo(device, os.getenv("USER"),
            [("delete", "192.0.2.4"), ("create", "192.0.2.8")])
```

Figure 8. *The ncclient script uses the candidate datastore and locking to modify the nameservers on a set of hosts identified on the command line.*

to document any such deviations and to convey the details to management applications. The deviate statement is used to formally define deviations, and the existence of deviations is announced via NETCONF's capability exchange mechanism.

TRANSLATIONS

The YANG language must exist in environments where a great number of languages and tools are already used. Hence, it is important to develop translation algorithms from YANG to other schema languages and from existing network management data modeling languages to YANG. Some translation algorithms are being considered during the design of the YANG language to ensure that feasible translation algorithms exist.

YANG can be translated into other XML schema languages, most notably XSD and RelaxNG. This allows developers to use standard XML tools as they see fit. The translation of YANG into RelaxNG is part of a bigger effort of defining translation rules of YANG into Document Schema Definition Languages (DSDL), of which RelaxNG is a part.

In order to reuse the large amount of existing SMIv2 definitions, a one-way translation of SMIv2 into YANG has been realized. A translation of YANG into SMIv2 was not considered since YANG is a much more powerful data modeling language, and a generic translation to SMIv2 would lead to either SMIv2 constraints being carried over to YANG or to translations to SMIv2 that are too clumsy to be used in practice.

IMPLEMENTATIONS

Several NETCONF implementations have been created and are being deployed. Major network device manufactures such as Juniper Networks, Ericsson, Cisco Systems, and Nortel ship NETCONF as part of their device software. Other companies such as Tail-f Systems, Netconf Central, SNMP Research, and Silicon and Software Systems license complete NETCONF development kits to device manufacturers. Several open source NETCONF implementations are under development. Efforts to develop interoperability test suites have started recently [9], and discovered implementation bugs as well as ambiguities in the protocol specification are being fleshed out.

Several commercial and open source implementations of the YANG data modeling language have been developed. Some YANG compilers support translation of YANG models to the YIN format, XSD, and RelaxNG. To facilitate access to SNMP data models through NETCONF, an SMIv2-to-YANG translation algorithm has been implemented as part of an open source SMIv2 toolset.

Finally, some high-level programming frameworks are being developed to make it easy to glue NETCONF into network-wide configuration and policy systems. A good example is the `ncclient` API for Python [10]. Figure 8 shows a Python script modifying the nameservers used by the set of hosts identified on the command line. The script uses the `candidate` datastore and protects itself against concurrent scripts by locking the `candidate` datastore.

RELATED WORK

The YANG language has been influenced by the design of the SMIng data modeling language [11]. Different from SMIng, YANG does not aim at being a protocol-independent language. Based on the experience with the SMIng approach [12], a design decision was taken very early to design YANG as a domain-specific NETCONF data modeling language.

An initial study of performance aspects of the NETCONF protocol can be found in [13]. While it has been acknowledged by the IETF working group that an access control subsystem for NETCONF is needed, standardization work in this area has not yet been started. Some early research in this area can be found in [14].

CONCLUSIONS

The design of NETCONF and YANG incorporates decades of experience from network device vendors, standardization bodies, implementers, and operators, and therefore has great potential to make network configuration management simpler, more effective, and more robust.

The NETCONF protocol addresses the requirements for configuration management protocols defined in [3] well and is therefore a good choice for this task. The pragmatic approach to layer NETCONF on top of a secure and reliable transport greatly simplifies the protocol. The development of the YANG data modeling language and its standardization in the IETF is progressing well. The availability of open source YANG tools has already motivated IETF working groups to use YANG for writing configuration data models, even though some aspects of the YANG language are still being finalized, and some toolkit vendors have already replaced their proprietary data modeling language with YANG.

While NETCONF and YANG provide a strong base technology for simpler, more effective, and more robust configuration management, additional cost savings will be achieved by the definition of standard configuration data models. This will be a longer-term effort because it requires identifying and agreeing on common subsets of configuration information that can be easily augmented with vendor-specific extensions for the vendor-specific features that differentiate their products. But even with just a small set of common configuration data models, NETCONF and YANG offer technology to deal with proprietary configuration data models in a much more cost-effective and robust way.

Even with just a small set of common configuration data models, NETCONF and YANG offers a technology to deal with proprietary configuration data models in a much more cost effective and robust way.

Acknowledgments

The work on this article was supported in part by the EC IST-EMANICS Network of Excellence (#26854).

References

[1] R. Enns, "NETCONF Configuration Protocol," Juniper Networks, RFC 4741, Dec. 2006.
[2] J. Case *et al.*, "Introduction and Applicability Statements for Internet Standard Management Framework," SNMP Research, Network Associates Laboratories, Ericsson, RFC 3410, Dec. 2002.
[3] J. Schönwälder, "Overview of the 2002 IAB Network Management Workshop," International University Bremen, RFC 3535, May 2003.
[4] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," SSH Communications Security Corp, Cisco Systems, RFC 4251, Jan. 2006.
[5] S. Chisholm and H. Trevino, "NETCONF Event Notifications," Nortel, Cisco, RFC 5277, July 2008.
[6] D. C. Fallside, "XML Schema Part 0: Primer Second Edition," W3C Rec., Oct. 2004.
[7] J. Clark and M. Makoto, "RELAX NG Specification," OASIS Committee Spec., Dec. 2001.
[8] M. Björklund, "YANG -A Data Modeling Language for NETCONF," Tail-f Systems, RFC (to appear), 2010.
[9] H. M. Tran, I. Tumar, and J. Schönwälder, "NETCONF Interoperability Testing," *Proc. 3rd Int'l. Conf. Autonomous Infrastructure, Mgmt., Sec. '09*, ser. LNCS, no. 5637, Springer, June 2009, pp. 83–94.
[10] S. Bhushan, H. M. Tran, and J. Schönwälder, "NCClient: A Python Library for NETCONF Clients," *Proc. IPOM '09*, *LNCS*, no. 5843, Venice, Springer, Oct. 2009, pp. 143–54.
[11] F. Strauß and J. Schönwälder, "SMIng — Next Generation Structure of Management Information," TU Braunschweig, IU Bremen, RFC 3780, May 2004.
[12] J. Schönwälder, "Protocol Independent Network Management Data Modeling Languages — Lessons Learned from the SMIng Project," *IEEE Commun. Mag.*, vol. 46, no. 5, May 2008, pp. 148–53.
[13] S.-M. Yoo, H. T. Ju, and J. W. Hong, "Performance Improvement Methods for NETCONF-Based Configuration Management," *Proc. APNOMS '06*, ser. LNCS, no. 4238, Busan, Korea, Springer, Sept. 2006, pp. 242–52.
[14] V. Cridlig, R. State, and O. Festor, "An Integrated Security Framework for XML-Based Management," *Proc. 9th IFIP/IEEE Int'l. Symp. Integrated Net. Management*, May 2005, pp. 587–600.

Biographies

JÜRGEN SCHÖNWÄLDER (j.schoenwaelder@jacobs-university.de) is an associate professor at Jacobs University Bremen, where he leads the computer networks and distributed systems research group. He is an active member of the IETF, where he has co-edited more than 25 network management related specifications and standards. He is currently serving as Chair of the NMRG of the Internet Research Task Force and Co-Chair of the ISMS Working Group of the IETF.

MARTIN BJÖRKLUND is chief software architect and co-founder of Tail-f Systems, and has more than a decade of network management systems experience working for companies such as Ericsson, Alteon, and Nortel. He serves as an Editor in the IETF's NETCONF and NETMOD Working Groups. He holds an M.Sc. in computer science from the Royal Institute of Technology in Stockholm.

PHIL SHAFER has worked for 12 years in the user interface and network management areas for Juniper Networks, creating the JUNOS UI, defining features of the command line interface, defining the XML API, and working to build a user experience second to none. He has worked to add features that simplify configuration and make life easier for users. He serves as an Editor in the IETF's NETMOD Work-