A Python Module for NETCONF Clients

Shikhar Bhushan

Computer Science
Jacobs University Bremen
Campus Ring 1
28759 Bremen
Germany

Type: Guided Research Proposal
Date: March 8, 2009
Supervisor: Prof. J. Schönwälder

Executive Summary

Networks have rapidly increased in size, complexity, and the number of services dependent on them. However, network management tools have not kept pace with this dizzying growth.

NETCONF is an answer to the problems of heterogeneity and scale inherent in computer networks. It is a new network management protocol, developed with broad participation of the industry. It provides secure and robust facilities for automating configuration operations. Network services providers have much to gain by the realization of economies around NETCONF.

In this document, I propose developing a free software library for NETCONF in the Python programming language. The deliverable will be modular and extensible. It will abstract protocol details and expose an intuitive, object-oriented Application Programming Interface (API) for the development of network management applications.

---

1 Summary

The Network Configuration protocol (NETCONF) is a network management protocol developed by the Netconf Working Group [1] of the IETF. It is a proposed internet standard since December 2006. [2, RFC 4741]

> [NETCONF] provides mechanisms to install, manipulate, and delete the configuration of network devices. It uses an Extensible Markup Language (XML)-based data encoding for the configuration data as well as the protocol messages. The NETCONF protocol operations are realized on top of a simple Remote Procedure Call (RPC) layer. [2]

NETCONF has quickly positioned itself as the superior standard for network configuration. There has been good industry adoption of the protocol, and most network equipment vendors have either implemented or plan to implement it. It is a significant improvement over proprietary command line interfaces commonly used for configuring network devices. Efforts are underway to create standard data models for NETCONF and data manipulated by it. [3] This will enable a degree of interoperability that has not been possible so far.

I herein propose the the development of a Python module for NETCONF clients. Python is already in common use as a scripting language in networked environments, as well as within the NETCONF community, e.g. EnSuite [4], Yang [5].

2 Statement and Motivation of Research

The Simple Network Management Protocol (SNMP) is an early network management standard has served important monitoring functions. However, its configuration facilities are rarely utilized in practice and users have preferred working with proprietary command line interfaces. Despite its name, SNMP is not a simple protocol to implement.

I find the following text, emanating from an internet draft of the Netmod Working Group, to be a fitting quote:

> Networks are increasing in complexity and capacity, as well as the density of the services deployed upon them. Uptime, reliability, and predictable latency requirements drive the need for automation.
>
> The problems with network management are not simple. They are complex and intricate. But these problems must be

> solved for networks to meet the stability needs of existing services while incorporating new services in a world where the growth of the networks is exhausting the supply of qualified networking engineers. We need to move from a CLI world into a world of automation, but that automation must be robust and trustworthy. [6]

The NETCONF protocol is designed for robust and trustworthy automation. It is a secure, connection-oriented protocol. Its use of XML as a data encoding language gives a "rich, flexible, hierarchical, standard representation of data that matches the needs of networking devices." [6]

The Netconf Working Group has focused its efforts on defining the base NETCONF standard, explicitly leaving out of its charter data modeling issues. The Netmod Working Group came about to fill in this gap, and has developed YANG, a language "to model configuration and state data manipulated by the NETCONF protocol, NETCONF remote procedure calls, and NETCONF notifications." [7]

The realization of NETCONF's promise would be greatly aided by a software library that makes it easy to create network management applications around it. Through this guided research project, I want to address this with an open source implementation in Python. My aim for this project is that it should be easy to write a simple script that deploys a configuration change across routers from different vendors, and at the same time possible to implement more complex applications like web interfaces and management consoles.

In the time span of this project, it will not be possible to incorporate data modeling features. My primary focus will be creating an intuitive, stable API for the base NETCONF specification set out in RFC 4741 [2].

It is worthwhile to note that there has already been a free software implementation of NETCONF in Python called `EnSuite` [4]. Besides a NETCONF server, `EnSuite` includes a client called `YencaP Manager`. However, this is geared towards a web-based management interface. I determined that adapting `YencaP Manager` towards this project's aims would require refactoring or rewriting major sections of the code, and consequently I propose to start from scratch with a cleaner architecture.

Proprietary implementations of NETCONF clients include a Java implementation from Tail-f Systems called `ConfM` and a Windows-only application called `NETCONF Client GUI` from Cisco Systems, Inc. The latter is intended for didactic purposes rather than use in production. Juniper Networks documents a Perl module `Net::Netconf::Manager` on its website, but download access is restricted to Juniper customers.

3 Planned Approach

Figure 1: NETCONF protocol overview [2]

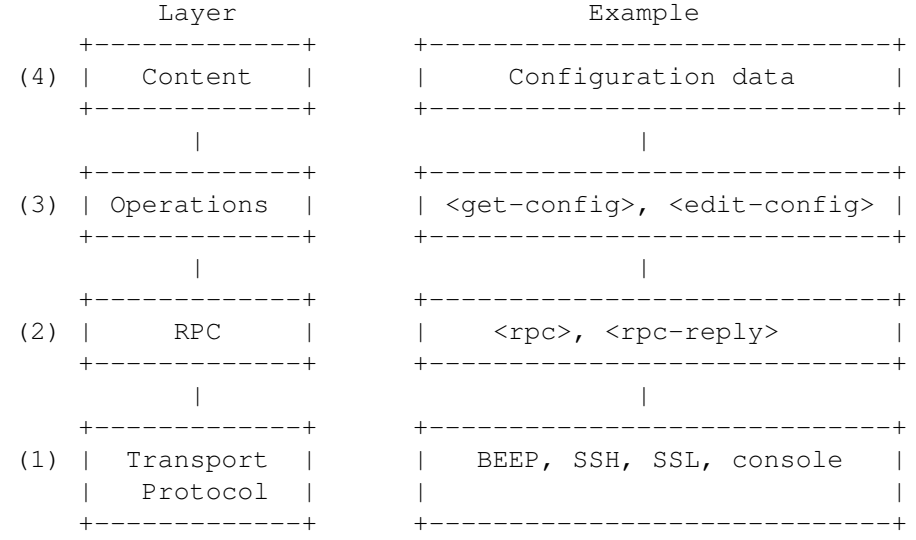

Figure 1 illustrates how NETCONF may be conceptually partitioned into logical layers. This is in fact an excellent architecture for our purpose. It mirrors the actual protocol and minimizes the degree of coupling between the different layers.

The `ncclient` module will be organized into three submodules based on these layers: `session`, `operations` and `content`.

The high-level API will start with the creation of a `Manager` object. This will provide syntactic sugar that abstracts how the different modules interact. For more direct control, it will be possible to compose the modules manually.

Code listing 1 and 2 are illustrative of the general API that I would like to work towards.

3.1 Capabilities

Capabilities introduce new protocol operations and modify existing operations, when mutually understood by server and client. In addition to the `:base` NETCONF capability, the capabilities listed in Table 1 will be supported. A plug-in manner to support additional capabilities will be investigated and implemented.

Code Listing 1: A simple example using `Manager`

```
import ncclient

config = ""
# using the new with..as syntax introduced in Python 2.5
with ncclient.Manager(host="10.0.0.1", user="x", passwd="x") as device:
    try:
        config = device.getConfig(source="running")
    except RPCError as e:
        print(e.message)
```

Code Listing 2: Manually composing different modules

```
import ncclient
from ncclient import session, operations, content

try:
    s = session.SSH(host="10.0.0.1", port=99)
    try:
        # by default use local key files / ssh agent
        s.connect(ncclient.capabilities)
        assert(":url" in session.peerCapabilities)
        op = operations.CopyConfig()
        op.params["source"] = "running"
        op.params["target"] = "https://user@example.com:passphrase/backup"
        op.responseHandler = content.ResponseHandler
        op.execute(s)
    except RPCError as e:
        pass # handle
except UnknownHost as e:
    pass # handle
finally:
    operations.CloseSession().execute(s)
```

3.2 Submodules

`session`

NETCONF is designed to be independent of session-layer and transport. Currently NETCONF has transport mappings defined for SSH [9, RFC 4742], SOAP [10, RFC 4743] and BEEP [11, RFC 4744]. Of these, a NETCONF server has to mandatorily implement SSH. [2] This, and the ubiquity of SSH, makes it the best choice for initial development.

Programmatically establishing and utilizing a SSH session would be preferable to forking a `ssh` subprocess, since this gives finer control over authentication and host verification. `paramiko` [12] is a SSH library for Python that was evaluated and found to be very suitable for this purpose. RFC 4742 [9] makes note of several security considerations which will be taken

Table 1: Planned capability support

Capability	Description
:writable-running [2]	<running/> configuration can be directly modified.
:candidate [2]	Utilize <candidate/> configuration database.
:confirmed-commit [2]	Server waits for a timeout period for client's confirmation before committing.
:rollback-on-error [2]	'All-or-nothing' edit mode.
:startup [2]	Distinct <startup/> configuration database.
:url [2]	URL permissible as source or target of operations.
:validate [2]	Request validation of configuration data.
:xpath [2]	Filtering using XPath expressions.
:notification [8]	NETCONF event notifications.
:interleave [8]	Interleave active notification subscription with other operations.

into account.

Extensibility with respect to other transport protocol mappings will be ensured by creating a base class that implements and exposes common functionality. Effort will be made to limit the protocol-specific API.

Only supporting blocking operation would greatly reduce the benefits of this library. It is important that the transport channel is used asynchronously to take advantage of NETCONF's pipelining support. Asynchronous operation is also required for supporting the :interleave capability. Therefore, all transport layer code will run in a separate thread. Doing this "right" will be challenging, but with the payoff that there will be a sound base for application development.

Functions

- Establish session: authentication, host verification, and capability exchange.
- Pipelining of requests.
- Distinguish incoming messages.

- Tearing down the session.

`operations`

This module corresponds to the RPC and Operations layers in Figure 1. Each NETCONF operation will correspond to an `Operation` object. All the operations specified in RFC 4741 [2] and RFC 5277 [8] will be implemented.

Functions

- Interact with session layer.
- Wrap operations as RPC requests taking into account available capabilities.
- Map response or notification to pertinent `Operation` object.

`content`

This module will have the most to do with XML processing. A suitable XML library remains to be determined. I expect that the Python standard library's facilities for XML should prove sufficient. Going this route will also reduce external dependencies.

As a rule of thumb, everything that involves looking into NECONF messages will be handled by this module. It will provide generic classes that may be extended.

Functions

- Interact with operations layer.
- Provide facilities for creating valid XML documents and parsing responses/notifications.

It is proposed that data modeling features be taken up at a future date once the ground for YANG [7] is set. This will be facilitated by the existence of a Python module for YANG that is actively developed. [5]

3.3 Licensing and Distribution

All code will be licensed under the BSD license. A permissive license was chosen to remove barriers for the use and future development of this library. The module will be packaged using the Python distribution utilities.

3.4 Testing

Each module will have unit tests that make use of the Python unit testing framework.

Interoperability testing is also important in light of the fact that there can be many different implementations of NETCONF servers. `ncclient` will be thoroughly tested against the Cisco, Juniper and Tail-f implementations.

3.5 Network-wide configuration

The above discussion has focused on configuring individual network devices. The true potential of NETCONF is realized in configuring entire networks of heterogeneous devices. To this end a `MultiManager` class will be implemented on top of the `ncclient` module. This will also serve as sample code of API usage.

4 Evaluation criteria

The deliverable is a Python module that makes it easy to develop configuration network management applications for NETCONF. Thus the criteria for evaluation is rather subjective.

Architecture Sound code organization and object-oriented design, exposure of an intuitive and flexible high-level API.

Code quality Idiomatic Python.

Distributability Conformance to standard packaging practice.

Documentation Quality of documentation and provision of sample code.

Extensibility Ease of implementing new functionality without requiring rewrites.

Standards-compliance With regard to requirements and recommendations of relevant standards - RFC 4741 [2], RFC 4742 [9], and RFC 5277 [8].

Testing Tested to work, what it works with, and test coverage.

5 Work plan

There are essentially 6 weeks available for the execution of the project. Here I describe how long I expect to spend on various aspects.

March 17 - 31, 2009 Implement and test `session` module.

April 1 - 10, 2009 Implement and test `operations` and `content` modules.

April 10 - 30, 2009 Integration, interoperability testing, `MultiManager`, and documentation.

References

[1] Network Configuration (netconf) Charter. `http://www.ietf.org/html.charters/netconf-charter.html`.

[2] R. Enns (Ed.). NETCONF Configuration Protocol. RFC 4741, Internet Engineering Task Force, December 2006.

[3] NETCONF Data Modelling Language (netmod) Charter. `http://www.ietf.org/html.charters/netmod-charter.html`.

[4] J. Bourdellon V. Cridlig, H. Abdelnur and R. State. A NetConf Network Management Suite: ENSUITE. In *5th IEEE International Workshop on IP Operations & Management*, volume 3751 of *Lecture Notes in Computer Science*, pages 152–161. Springer, October 2005.

[5] pyang - An extensible YANG validator and converter in Python. `http://code.google.com/p/pyang/`.

[6] P. Shafer. An Architecture for Network Management. Internet draft, Internet Engineering Task Force, March 2009. Work in progress.

[7] YANG - A data modeling language for NETCONF. Internet draft, Internet Engineering Task Force, March 2009. Work in progress.

[8] S. Chisholm and H. Trevino. NETCONF Event Notifications. RFC 5277, Internet Engineering Task Force, July 2008.

[9] M. Wasserman and T. Goddard. Using the NETCONF Configuration Protocol over Secure SHell (SSH). RFC 4742, Internet Engineering Task Force, December 2006.

[10] T. Goddard. Using NETCONF over the Simple Object Access Protocol (SOAP). RFC 4742, Internet Engineering Task Force, December 2006.

[11] E. Lear and K. Crozier. Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP). RFC 4744, Internet Engineering Task Force, December 2006.

[12] paramiko - SSH2 protocol for Python. `http://www.lag.net/paramiko/`.