Building NETCONF-enabled Network Management Systems with libnetconf

Radek Krejčí
CESNET, z.s.p.o.
Zikova 4, 160 00 Praha 6, Czech Republic
rkrejci@cesnet.cz

***Abstract*—NETCONF is proposed as a new standard protocol for configuration and management of network devices. NETCONF provides standardized way to configure heterogeneous networks. Currently, one of the obstacles to its widespread adoption is a lack of programming tools allowing developers to easily add NETCONF protocol support into the configuration tools and network devices. *libnetconf* library provides such functionality.**

This paper presents possibilities of using *libnetconf* for a development of network management applications and network device controllers supporting NETCONF protocol. We describe NETCONF functions provided by *libnetconf* as well as guidelines for utilizing *libnetconf* library to develop network management applications.

***Index Terms*—libnetconf, NETCONF, network configuration, network management**

I. Introduction

Utilization of present computer networks is more and more diversified. Demands of the users make networks heterogeneous and complex. Therefore, a simple and efficient management of various network devices is a crucial task for network operators.

Discussion about strengths and weaknesses of the specific network management technologies can be found in RFC 3535 [1]. According to the conclusions of this document, the Internet Engineering Task Force (IETF) set up the network configuration (NETCONF) Working Group to focus on standardization of configuration management mechanisms. The main outcome of the group is the NETCONF protocol specification [2].

This paper presents *libnetconf* – an open source[1] library implementing the NETCONF protocol for the GNU/Linux. Allowing developers to simply add the NETCONF protocol support into their device or system is the main benefit of the library. *libnetconf* allows developers to concentrate directly on the device configuration.

The rest of the paper is organized as follows. At first, the NETCONF protocol is introduced and all currently available open source implementations are briefly presented. Next, the *libnetconf* library is described and the example implementations of NETCONF server and client are briefly presented.

[1]Available from http://libnetconf.googlecode.com under BSD license.

II. NETCONF Protocol

NETCONF conceptually stems from JUNOScript. Both are based on a lightweight remote procedure call (RPC) mechanism using messages encoded in XML. NETCONF uses layered architecture to separate configuration data and underlying transmission protocols (see Figure 1). Configuration data are stored in the datastore reflecting the current device parameters.

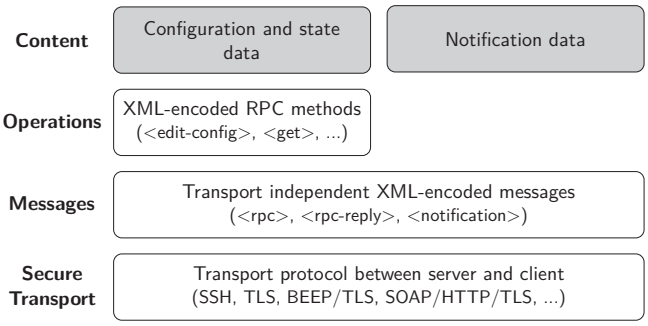

Fig. 1. The NETCONF Protocol Layers [2]

NETCONF specification defines requirement for the transport protocols used with NETCONF. However, Secure Shell (SSH) [3] is specified as the mandatory one. [2, Section 2]

Next two layers form the NETCONF XML-encoded RPCs. These RPCs represents NETCONF operations to manipulate configuration data in the datastore of the targeted device or system. Exceptionally, an RPC can affect the NETCONF session or the server itself.

Basic NETCONF functionality can be extended using newly defined capabilities. They allow modification of current operations, definition of new ones or addition of some other functions into both communication participants. During the NETCONF session establishment, server provides the set of supported capabilities that can be used during further communication. As an example, RFC 5277 [4] defines NETCONF Event Notifications as a NETCONF capability providing asynchronous message delivery mechanism for NETCONF.

III. Related Work

There are several implementations of the NETCONF protocol[2]. Unfortunately, some of them are available only as

[2]The list maintained by the NETCONF Working Group is available from http://tools.ietf.org/wg/netconf/trac/wiki.

978-3-901882-50-0

commercial products or they are out of date and not maintained by the authors anymore.

Netconf4Android and ncclient implement only client side of the NETCONF protocol following obsoleted RFC 4741. EnSuite and YENCA provide server side implementation, but they are out of date and poorly documented. The last notable open source implementation is Yuma. It provides client as well as server side implementation of the NETCONF protocol following the last RFC 6241 [2].

Yuma offers a plugin mechanism to add modules into the server and control a specific device or a system. Thus, it is a complete NETCONF server with possibility to add functionality to configure specific device. In contrast to Yuma, *libnetconf* allows developer to add NETCONF capability into their systems and devices.

IV. NETCONF Library

The main idea of the *libnetconf* library is to separate and hide NETCONF functionality into the library itself. While developers are working on the user configuration interface, they should not be burdened with the communication protocol between their application and the controlled system. Similarly, device and system developers should focus on applying configuration changes instead of managing configuration data and communication with clients. *libnetconf* provides NETCONF functions for building NETCONF clients as well as NETCONF servers.

Yuma requires running its complex NETCONF server on the controlled device. In contrast, using *libnetconf* requires building whole new server. On the other hand, such server can be very simple or quite complex according to the specific requirements of the controlled device. This potentially decreases resource requirements for the controlled device, which can be highly valuable in case of embedded devices.

libnetconf is based on authors' experiences with developing standalone NETCONF remote configuration system (Netopeer[3]). Currently, there is a new generation of the Netopeer software being developed using *libnetconf*. Besides this usage, there are client and server examples providing details of using specific features of the library. These examples are available as part of the *libnetconf* source codes. Also a complete Doxygen documentation is available for all library functions.

The NETCONF protocol can be layered on several transport protocols, but SSH is defined as the mandatory one. *libnetconf* provides NETCONF connections only over SSH using libssh2[4] on the client side and SSH Subsystem mechanism on the server side. It means, that NETCONF server side application is launched by an SSH daemon as its Subsystem [5, Section 5.7] and *libnetconf* accepts incoming NETCONF connection through the SSH daemon.

A. NETCONF Protocol Versions

RFC 6241 introduces several changes to the NETCONF protocol. These changes required increment of the protocol version to 1.1. However, the main difference between NETCONF 1.0 and 1.1 is in a framing mechanism when using NETCONF over SSH. The new Chunked Framing mechanism [3, Section 4.2] is defined for NETCONF version 1.1. *libnetconf* supports both the NETCONF versions.

B. System Architectures

libnetconf must be applicable in case of very simple applications as well as in case of very complex feature-rich applications providing NETCONF functionality. Using *libnetconf* in a NETCONF client is quite straightforward.

However, NETCONF servers can be implemented in very different ways. Basically, there are two approaches depicted in Figure 2 and described bellow. *libnetconf* is applicable in both of them.

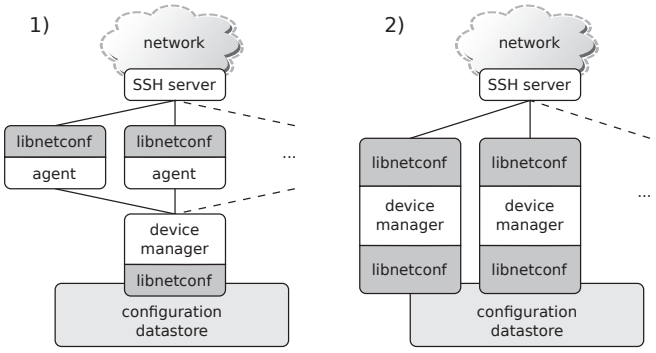

Fig. 2. NETCONF server architectures using *libnetconf*: 1) Multi-level architecture; 2) Single-level architecture.

1) The Multi-level Architecture: In this case, the device control facility is separated into a single autonomous process called *device manager*. This way, the problem of concurrency in accessing the controlled device is avoided. The NETCONF *agents* accept incoming NETCONF connections and then pass NETCONF messages to the *device manager*. *Agents* are invoked automatically by the SSH daemon using the SSH Subsystem mechanism while the *device manager* is running as a system daemon. For the inter-process communication between the *agents* and the *device manager* we recommend to use D-Bus[5] that serialize messages from multiple *agents*. *libnetconf* provides functions to (de-)serialize content of the NETCONF messages which simplifies passing NETCONF messages between the *agents* and the *device manager*.

2) The Single-level Architecture: In contrast to the previous case, *agents* are integrated into the *device manager* process. There is no persistently running system daemon. New instance of the *device manager* is invoked by the SSH daemon for each incoming NETCONF connection. The main challenge of this approach is a simultaneous access to the shared resources. *Device manager* itself has to solve a concurrent access to the controlled device from its multiple instances. *libnetconf* deals with a simultaneous access to the shared configuration datastore.

[3]Not maintained but still available from http://netopeer.googlecode.com.

[4]Available from www.libssh2.org.

[5]Message bus system, see www.freedesktop.org/wiki/Software/dbus

C. Datastores

libnetconf provides implementation of the NETCONF datastores. Internally, *libnetconf* defines the interface to add various types of datastore implementations in a future. Currently, there are two datastore implementations: *1)* using standard files and *2)* storing no data. The second one is used when all data in the device configuration model are defined as status data. Datastore implementation is transparent for the NETCONF server application accessing the datastore.

libnetconf internally deals with a simultaneous access to the datastore and avoids interleaving of different datastore operations. Thus, datastore operations seem to be atomic.

D. Supported NETCONF Capabilities

NETCONF capabilities extend functionality of the base NETCONF protocol. *libnetconf* divides them into the groups of basic and other capabilities. The basic group includes all capabilities defined in RFC 6241 (Writable-Running*, Candidate configuration*, Confirmed Commit, Rollback-on-Error, Validate, Distinct Startup*, URL and XPath capabilities). In the *libnetconf* client side API, all these capabilities are supported directly by the functions to create NETCONF RPC messages. Server side implementation currently supports only basic capabilities marked with asterisk in the list.

NETCONF allows definition of new capabilities in the future. Furthermore, any capability can specify modification to the existing NETCONF operations. This represents nontrivial issue for the design of the API. To avoid changes of the current *libnetconf* API with any newly supported capability, non-basic capabilities are handled by the special variadic function *nc_rpc_capability_attr()* modifying previously created RPC message. The function accepts a request code parameter specifying how the message should be changed. Request codes correspond to the specific modification of a particular capability. If an extra parameter for such change is required, the function accepts it as a variadic parameter. The list of supported request codes can be expanded with newly supported capabilities modifying the existing operation. This way a support for another NETCONF capability can be added into the *libnetconf* library with preserving backward compatibility.

Currently, besides the mentioned basic capabilities, *libnetconf* supports the following additional capabilities in both client and server side API.

1) With-defaults[6]*:* This capability changes processing and displaying of the default configuration values defined in the configuration data model. There are several with-defaults modes of the NETCONF server behavior. This mode can be selected via the *libnetconf* Server API. On the client side, application can modify RPC message using *nc_rpc_capability_attr()* to force specific with-defaults mode supported by the NETCONF server.

[6]Defined in RFC 6243 [6].

2) Event Notifications[7]*:* NETCONF basically uses synchronous message delivery – clients wait for answers to their RPC requests. This capability adds support of the asynchronous messages (notifications) to the NETCONF protocol. It allows server to announce an event occurrence to the clients that expressed their interest.

The *Event Notifications* is a specific capability since it does not only modify or add a new operation, but it fundamentally changes communication rules of the NETCONF protocol. The support of this capability had to be taken into account during the initial design of the library. Due to this capability, *libnetconf* internally uses separated message queues for the incoming event notifications and replies to the sent RPC requests. NETCONF client have to use separated functions to retrieve notification and RPC reply from the *libnetconf*'s message queues. These functions can be called as blocking, timeouted or non-blocking.

E. Concurrent Processing in libnetconf

Multi-threading support is a significant requirement for present applications. *libnetconf* allows concurrent access from multiple threads to the selected library objects such as NETCONF session or messages. Furthermore, due to providing statistical information specified by the NETCONF Monitoring module [7], *libnetconf* shares some generic information about active sessions between all processes that use *libnetconf*.

V. Using *libnetconf*

A. libnetconf API

libnetconf functions are divided into the following groups dealing with handling of basic objects provided by the library.

session: Functions to connect to the NETCONF server, accept incoming connection and get information about the established NETCONF session.

rpc: Functions to create, modify, send, receive and parse NETCONF RPC messages.

rpc-reply: Functions to create, modify, send, receive and parse NETCONF RPC reply messages.

event notifications: Functions to handle notification streams, enroll a new event and to create, send, receive and parse NETCONF notification messages.

datastore: Functions to handle NETCONF configuration datastores.

generic: Generic *libnetconf* functions.

B. libnetconf Client

A simplified scheme of the *libnetconf* client workflow is depicted on the left side of the Figure 3. After the application initiation, *libnetconf* is initiated and client can connect via SSH to the specified host using *libnetconf* functions. Before connecting, client is allowed to set up preferences of the SSH authentication methods[8]. These functions are included in *session* group.

[7]Defined in RFC 5277 [4].

[8]Interactive authentication, password authentication and authentication using SSH keys are supported.

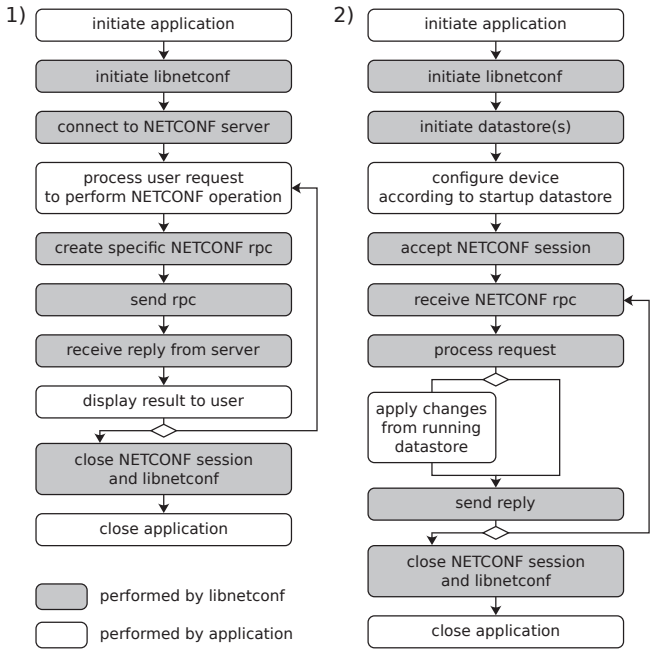

Fig. 3. Simplified workflow of the: 1) *libnetconf* client; 2) *libnetconf* server.

As the next step, client has to take request from the user. This part of work is out of the *libnetconf* scope.

Then the client creates the required NETCONF RPC message using *libnetconf* functions from the *rpc* group. Besides the set of basic NETCONF RPC operations that can be created directly by the functions provided by *libnetconf*, there is also a generic function that allows caller to create non-standard NETCONF RPC request. Created message is not connected with the specific NETCONF session and can be used multiple times in one or more NETCONF sessions.

Created RPC request is sent through the specified NETCONF session to the server. Then client receives the reply that can be processed by several functions provided by *libnetconf* in *rpc-reply* group.

After presenting results to the user, client processes another user request or closes NETCONF session and terminates.

C. libnetconf Server

The right side of the Figure 3 depicts a simplified workflow of a *libnetconf* server with the single-level architecture. The server is invoked as an instance of the netconf SSH Subsystem. After the application and *libnetconf* initialization, server gains access to the NETCONF datastore(s). If this is the first instance of the server and the device is not initiated, the server should apply configuration settings according to the NETCONF startup datastore. Now, the server is ready to accept incoming NETCONF connection that invoked the application as the netconf SSH Subsystem.

libnetconf takes care of negotiation of used NETCONF protocol version and all other things needed to establish the NETCONF session. Since now, the server is able to receive and processes NETCONF RPC requests using *libnetconf* functions. Furthermore, *libnetconf* automatically applies requested operations to the NETCONF datastores. If the requested operation targets the *running* datastore, the server should reflect performed changes to the controlled device. This operation is out of the *libnetconf* scope. When all requested operations are performed, a reply containing acknowledgment, requested data or error description is sent back to the client. This process is repeated until the session termination is required or until some error occurs.

D. Example Implementation

Example NETCONF server and client are part of the *libnetconf* source codes. Despite the simplicity, example client can be used as a regular command-line NETCONF client. Example NETCONF server demonstrates usage of the *libnetconf* library as described above.

VI. Conclusion

In this paper, we describe usage of *libnetconf* as an implementation of the NETCONF protocol – the IETF standard protocol for the network configuration. We provide guidelines for the implementation of NETCONF-enabled network management applications using *libnetconf*.

Judging from the feedback from the NETCONF community, the *libnetconf* library is well accepted. A library implementing the NETCONF protocol is supposed to force the integration of the NETCONF protocol in various management systems. The integration is made even easier by using C programming language, which provides a solid base for bindings in a wide range of programming languages. In November 2012, *libnetconf* based applications successfully took part in the IETF NETCONF interoperability testing.

Acknowledgment

This material is based upon work supported by the "CESNET Large Infrastructure" project LM2010005 funded by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] J. Schoenwaelder, "Overview of the 2002 IAB Network Management Workshop," RFC 3535 (Informational), Internet Engineering Task Force, May 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3535.txt

[2] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," RFC 6241 (Proposed Standard), Internet Engineering Task Force, Jun. 2011. [Online]. Available: http://tools.ietf.org/html/rfc6242

[3] M. Wasserman, "Using the NETCONF Protocol over Secure Shell (SSH)," RFC 6242 (Proposed Standard), Internet Engineering Task Force, Jun. 2011. [Online]. Available: http://tools.ietf.org/html/rfc6242

[4] S. Chisholm and H. Trevino, "Netconf event notifications," RFC 5277 (Proposed Standard), Internet Engineering Task Force, Jul. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5277.txt

[5] D. J. Barrett and R. E. Silverman, *SSH, The Secure Shell: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2001.

[6] A. Bierman and B. Lengyel, "With-defaults capability for netconf," RFC 6243 (Proposed Standard), Internet Engineering Task Force, Jun. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6243.txt

[7] M. Scott and M. Bjorklund, "YANG Module for NETCONF Monitoring," RFC 6022 (Proposed Standard), Internet Engineering Task Force, Oct. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc6022.txt