NCClient: A Python Library for NETCONF Client Applications

Shikhar Bhushan, Ha Manh Tran, Jürgen Schönwälder

JACOBS
UNIVERSITY

IPOM 2009, Venice, 2009-10-30

Support: EU IST-EMANICS Network of Excellence (#26854)

Outline of the Talk

NETCONF in a Nutshell

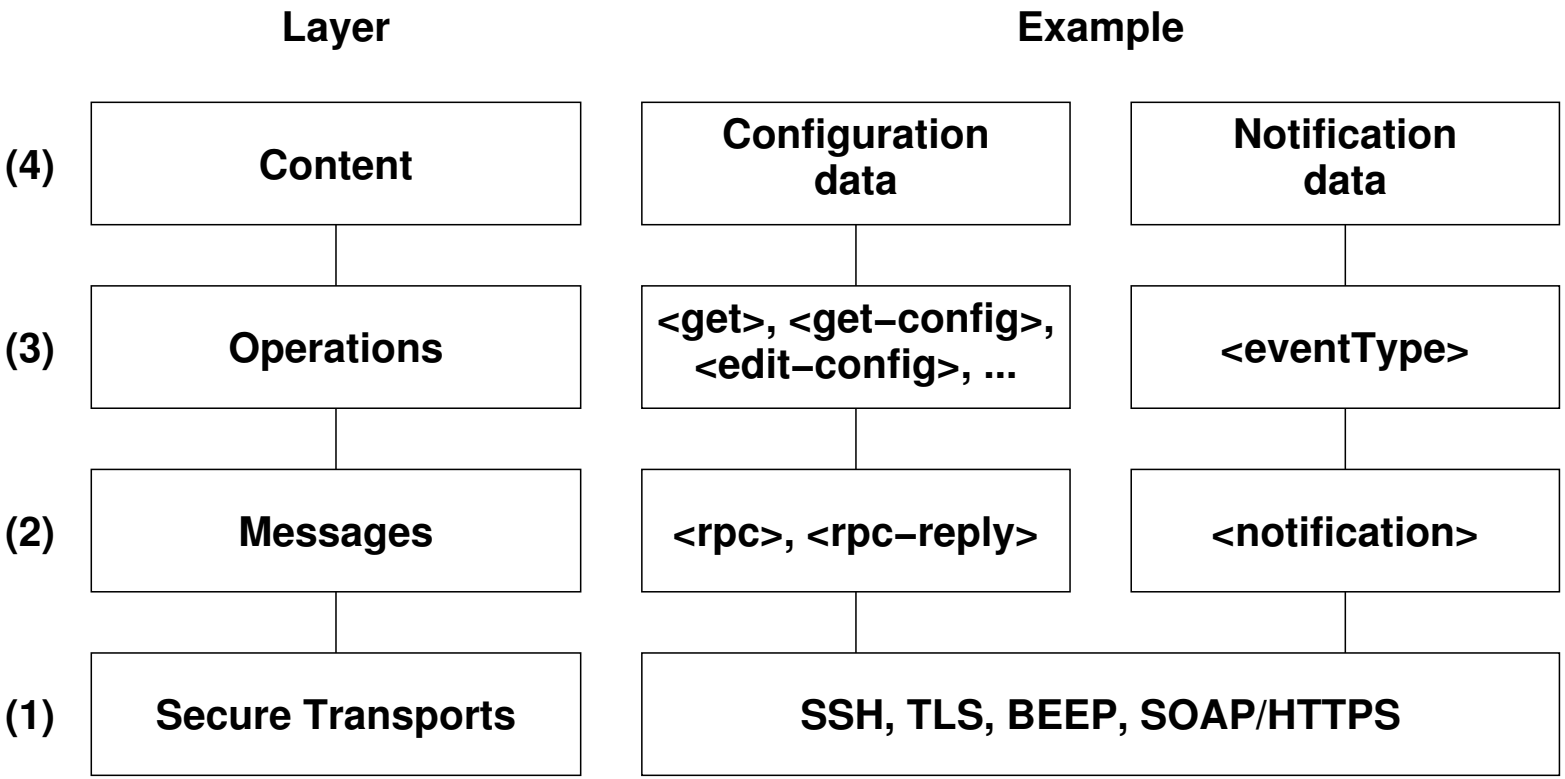
What is NETCONF?

NETCONF is a network management protocol specifically designed to support configuration management

NETCONF provides the following features:

- distinction between configuration and state data
- multiple configuration datastores (running, startup, ...)
- support for configuration change transactions
- configuration testing and validation support
- selective data retrieval with filtering
- streaming and playback of event notifications
- extensible remote procedure call mechanism

NETCONF Layers (do not trust RFC 4741)



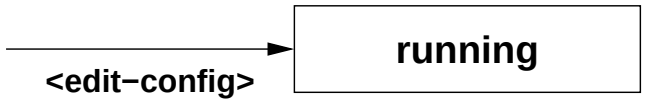
- George: Think CMIP of 21st century ;-)

NETCONF Operations

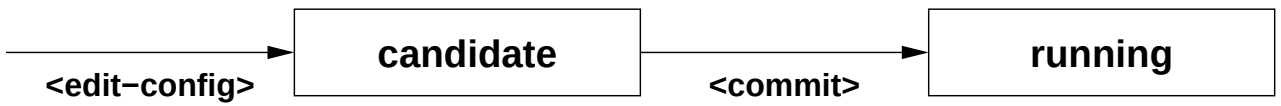
Operation	**Arguments**
get-config	source [filter]
edit-config	target [default-operation] [test-option] [error-option] config
copy-config	target source
delete-config	target
lock	target
unlock	target
get	[filter]
close-session	
kill-session	session-id
discard-changes	
validate	source
commit	[confirmed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

Transaction Models

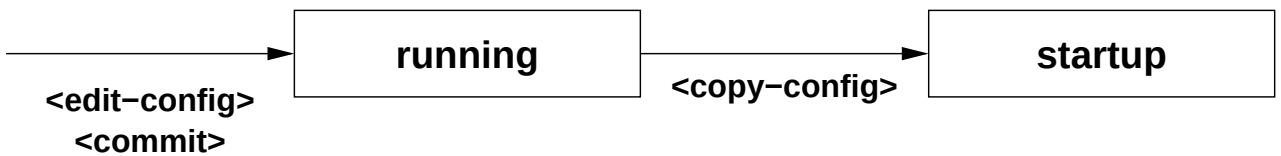
Direct Model



Candidate Model (optional)



Distinct Startup Model (optional)



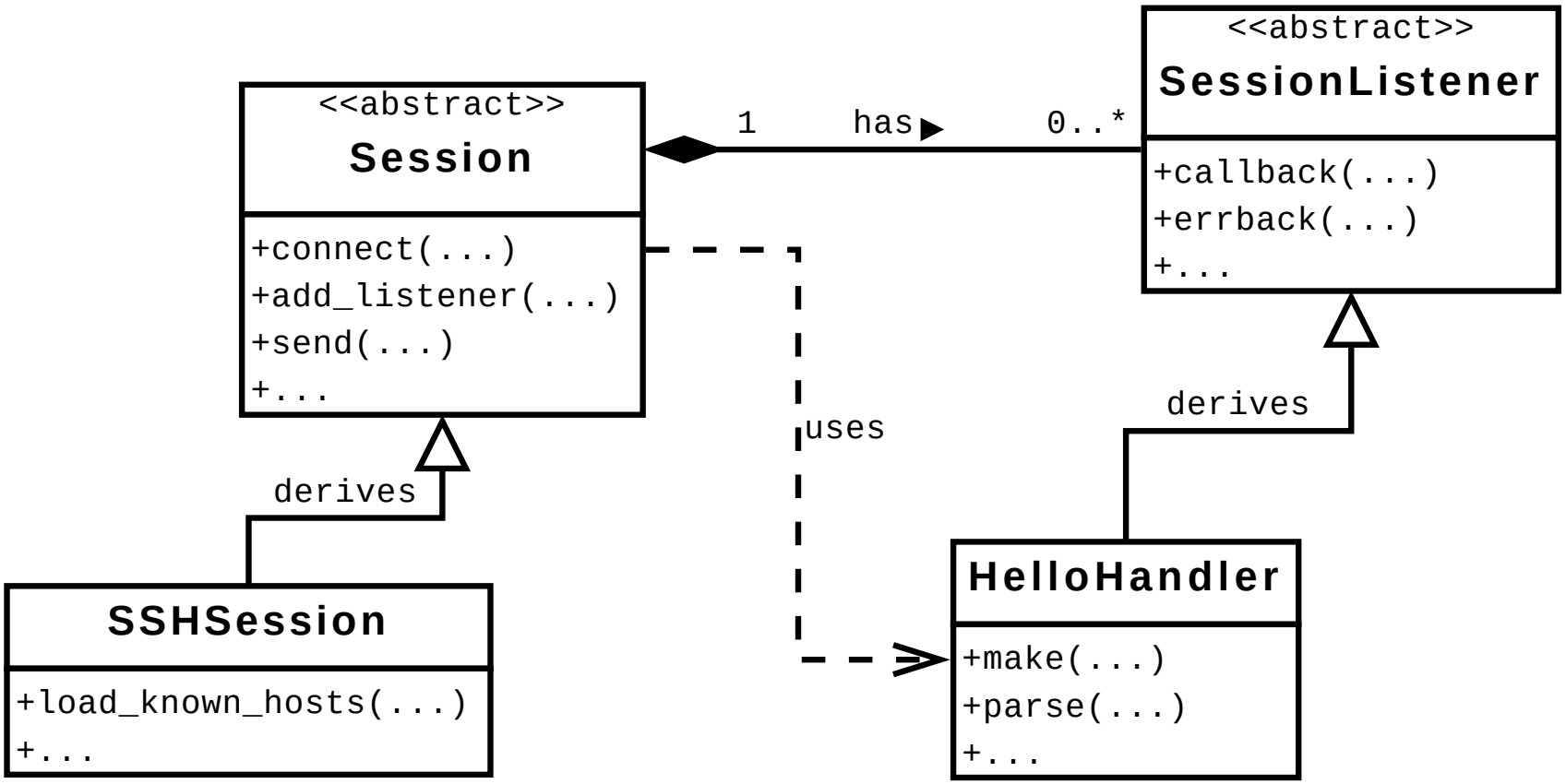
- Some operations (`edit-config`) support different error behaviours, including rollback behaviours

NCClient Sales Pitch

What is NCClient?

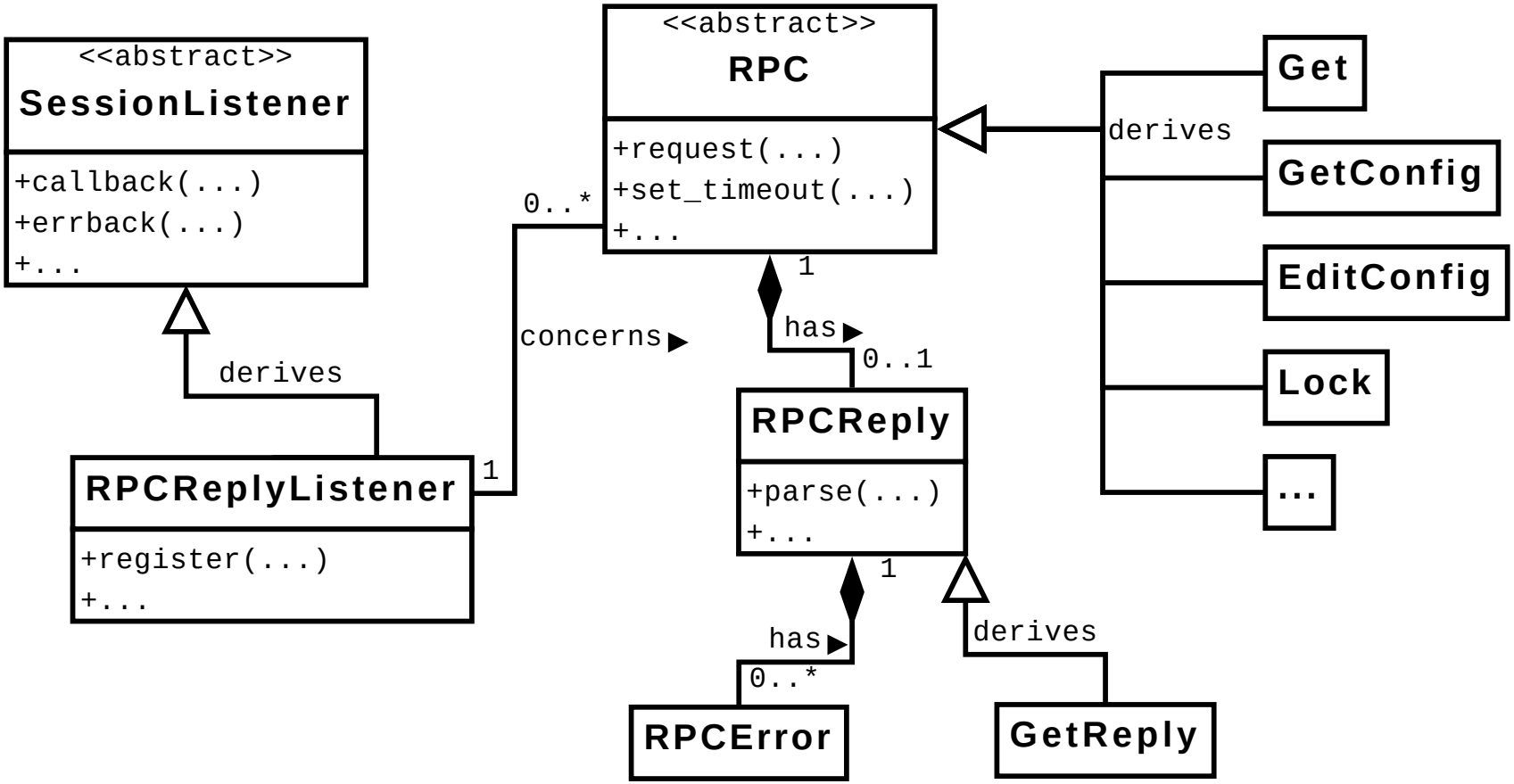
- high-level Python API for NETCONF
- extensibility to accomodate future protocol extensions
  - additional transports
  - additional protocol capabilities
  - additional remote procedure calls
- request pipelining
- asynchronous and synchronous RPC calls
- robustness through proper error/exception handling
- thread safety
- exploit Python programming language features
  - context managers for sessions and locks
  - exception handling
- open source (http://code.google.com/p/ncclient)

UML Diagram for Sessions and Transports



- Future transports can be plugged into NCClient by deriving additional classes from `Session`

UML Diagram for RPCs and Operations



- Future operations can be plugged into NCClient by deriving additional classes from `RPC`

Supported Capabilities

Capability	Supported
:writable-running	√
:candidate	√
:confirmed-commit	√
:rollback-on-error	√
:startup	√
:url	√
:validate	√
:xpath	√
:notification	
:interleave	

Interoperability Tests

Operation	Tail-f	Cisco	Netopeer
get-config	√	√	√
edit-config	√	√	√
copy-config	√	√	√
delete-config	√	√	√
lock	√	√	√
unlock	√	√	√
get	√	√	√
close-session	√	√	√
kill-session	√	√	√
discard-changes	√		
validate	√		
commit	√		

Examples / Demonstration

```
#! /usr/bin/env python2.6
#
# Connect to the NETCONF server passed on the command line and
# display their capabilities. This script and the following scripts
# all assume that the user calling the script is known by the server
# and that suitable SSH keys are in place. For brevity and clarity
# of the examples, we omit proper exception handling.
#
# $ ./nc01.py broccoli

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user):
    with manager.connect(host=host, port=22, username=user) as m:
        for c in m.server_capabilities:
            print c

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"))
```

```
#! /usr/bin/env python2.6
#
# Retrieve the running config from the NETCONF server passed on the
# command line using get-config and write the XML configs to files.
#
# $ ./nc02.py broccoli

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user):
    with manager.connect(host=host, port=22, username=user) as m:
        c = m.get_config(source='running').data_xml
        with open("%s.xml" % host, 'w') as f:
            f.write(c)

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"))
```

```
#! /usr/bin/env python2.6
#
# Retrieve a portion selected by an XPATH expression from the running
# config from the NETCONF server passed on the command line using
# get-config and write the XML configs to files.
#
# $ ./nc03.py broccoli "aaa/authentication/users/user[name='schoenw']"

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user, expr):
    with manager.connect(host=host, port=22, username=user) as m:
        assert(":xpath" in m.server_capabilities)
        c = m.get_config(source='running', filter=('xpath', expr)).data_xml
        with open("%s.xml" % host, 'w') as f:
            f.write(c)

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2])
```

```
#! /usr/bin/env python2.6
#
# Create a new user to the running configuration using edit-config
# and the test-option provided by the :validate capability.
#
# $ ./nc04.py broccoli bob 42 42

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user, name, uid, gid):
    snippet = """<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
        <authentication> <users> <user xc:operation="create">
        <name>%s</name> <uid>%s</uid> <gid>%s</gid>
        <password>*</password> <ssh_keydir/> <homedir/>
      </user></users></authentication></aaa></config>""" % (name, uid, gid)

    with manager.connect(host=host, port=22, username=user) as m:
        assert(":validate" in m.server_capabilities)
        m.edit_config(target='running', config=snippet,
                      test_option='test-then-set')

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2], sys.argv[3], sys.argv[4])
```

```
#! /usr/bin/env python2.6
#
# Delete an existing user from the running configuration using
# edit-config and the test-option provided by the :validate
# capability.
#
# $ ./nc05.py broccoli bob

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

def demo(host, user, name):
    snippet = """<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
        <authentication> <users> <user xc:operation="delete">
        <name>%s</name>
      </user></users></authentication></aaa></config>""" % name

    with manager.connect(host=host, port=22, username=user) as m:
        assert(":validate" in m.server_capabilities)
        m.edit_config(target='running', config=snippet,
                      test_option='test-then-set')

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2])
```

```
#! /usr/bin/env python2.6
#
# Delete a list of existing users from the running configuration using
# edit-config; protect the transaction using a lock.
#
# $ ./nc06.py broccoli bob alice

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

template = """<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
  <authentication> <users> <user xc:operation="delete">
  <name>%s</name> </user></users></authentication></aaa></config>"""

def demo(host, user, names):
    with manager.connect(host=host, port=22, username=user) as m:
        with m.locked(target='running'):
            for n in names:
                m.edit_config(target='running', config=template % n)

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2:])
```

```
#! /usr/bin/env python2.6
#
# Delete a list of existing users from the running configuration using
# edit-config and the candidate datastore protected by a lock.
#
# $ ./nc07.py broccoli bob alice

import sys, os, warnings
warnings.simplefilter("ignore", DeprecationWarning)
from ncclient import manager

template = """<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <aaa xmlns="http://tail-f.com/ns/aaa/1.1">
  <authentication> <users> <user xc:operation="delete">
  <name>%s</name> </user></users></authentication></aaa></config>"""

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

if __name__ == '__main__':
    demo(sys.argv[1], os.getenv("USER"), sys.argv[2:])
```

Conclusions

Conclusions

Recommendations

- Stop hacking nasty expect scripts or the like.
- Use Python and NCClient — it is cool and fun.
- Student assignments utilizing NCClient are a nice idea.
- If you use NCClient, please cite our IPOM 2009 paper. ;-)

Future Work

- Support for recent protocol extensions
  (e.g., fine grained locking, data model retrieval)
- High-level API supporting network-wide operations
- Generate RPC stub classes from YANG data models
- ...

References

R. Enns.

NETCONF Configuration Protocol.
RFC 4741, Juniper Networks, December 2006.

M. Wasserman and T. Goddard.

Using the NETCONF Configuration Protocol over Secure SHell (SSH).
RFC 4742, ThingMagic, ICEsoft Technologies, December 2006.

H. M. Tran, I. Tumar, and J. Schönwälder.

NETCONF Interoperability Testing.
In *Proc. of the 3rd International Conference on Autonomous Infrastructure, Management and Security (AIMS 2009)*, number 5637 in LNCS, pages 83–94. Springer, June 2009.

S. Bhushan, H. M. Tran, and J. Schönwälder.

NCClient: A Python Library for NETCONF Clients.
In *Proc. IPOM 2009*, number 5843 in LNCS, Venice, October 2009. Springer.