



Lab Session: NETCONF

by J. Schönwälder, H. Tran, V. Marinov
Jacobs University Bremen, Germany

Problem N.1: connecting to the NETCONF routers

(5 minutes)

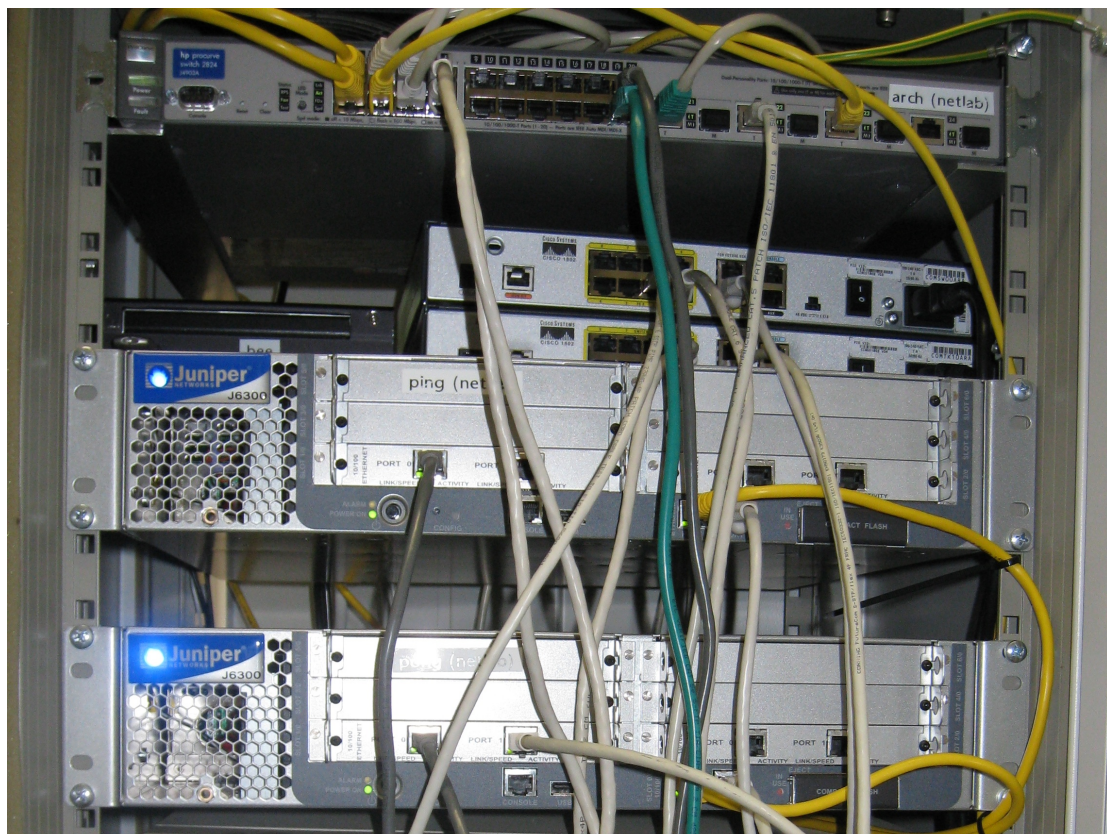
The NETCONF experiments will be done on two Juniper J6300 / Junos 9.0R1.10 routers called `ping` and `pong` and two Cisco C1802 / IOS 12.4 routers called `alex` and `rolf`. To access the routers, you first have to login to `meat.eecs.jacobs-university.de`:

- a) Log into `meat.eecs.jacobs-university.de` using

```
ssh <username>@meat.eecs.jacobs-university.de
```

- b) From `meat`, you can run the `netconf` shell script to connect to the routers:

```
netconf <username>@<routername>
```



Problem N.2: *capability exchange and graceful close*

(10 minutes)

Write a valid `hello` message that is accepted by both Juniper and Cisco routers and send a `close-session` operation to close the session. Answer the following questions:

- a) What are the capabilities announced by the two different implementations?
 - b) Write a valid `hello` message that is accepted by both Juniper and Cisco routers.
 - c) What is the purpose of the `session-id`?
 - d) Write a `close-session` operation to gracefully close the session.
-

Problem N.3: *retrieving configuration*

(10 minutes)

- a) Write a `get-config` operation that retrieves the running configuration of the Juniper and Cisco routers.
 - b) What is the major difference between the two implementations in terms of the representation of configuration information?
 - c) For the Juniper routers, write a `get-config` operation that retrieves only the `ospf` configuration from the running configuration.
-

Problem N.4: *lock, validate, commit, unlock*

(10 minutes)

This problem is to be solved on Juniper routers.

- a) Write a `lock` operation to lock the candidate configuration. What happens if the lock can't be acquired?
 - b) Write an `unlock` operation to unlock the candidate configuration.
 - c) Write a `discard-changes` operation to copy the running configuration to the candidate configuration.
 - d) Write a `validate` operation to validate the candidate configuration.
 - e) Write a `commit` operation to commit the candidate configuration to the running configuration
-

Problem N.5: *editing configuration*

(10 minutes)

This problem is to be solved on Juniper routers. Please keep the locking times short since we only have a very small number of routers.

- a) Lock the candidate configuration and discard any changes.
 - b) Assign a random IP address out of the 10.0.0.0/8 address block to one of the physical interfaces fe-2/0/0 or fe-2/0/1 on either ping or pong using the `edit-config` operation.
 - c) Check the result by retrieving the candidate configuration using the `get-config` operation.
 - d) Discard your changes and unlock the candidate configuration.
-

Problem N.6: *illegal input and corner cases*

(10 minutes)

- a) How do the Juniper and the Cisco implementations react to an unknown operation?
 - b) What is a fundamental difference how messages are processed by the two implementations?
 - c) What happens if the `rpc` element contains additional attributes or lacks the `message-id` attribute?
 - d) What happens if the XML declaration is missing?
-

Problem N.7: configure an OSPF routing topology

(20 minutes)

Using NETCONF, configure the simple OSPF (Open Shortest Path First) [1] routing topology shown in Figure 1.

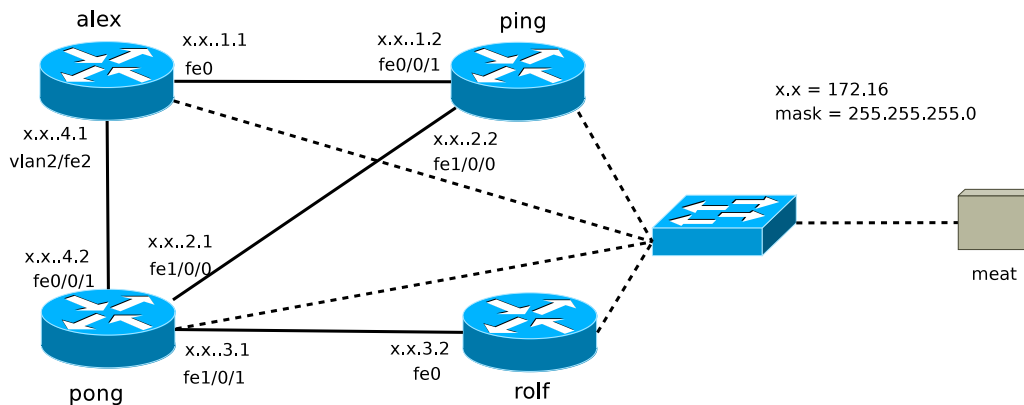


Figure 1: A topology including routers with FastEthernet (fe) and Vlan (vlan) interfaces

Below is the CLI configuration of the routers. You can find descriptions how to translate CLI commands to NETCONF operations in [2, 3].

- Router alex:

```
interface FastEthernet0
ip address 172.16.1.1 255.255.255.0

interface Vlan2
description OSPF
ip address 172.16.4.1 255.255.255.0

interface FastEthernet2
switchport access vlan 2

router ospf 1
network 172.16.0.0 0.0.255.255 area 0.0.0.0
```

- Router rolf:

```
interface FastEthernet0
bandwidth 10000
ip address 172.16.3.2 255.255.255.0

router ospf 1
auto-cost reference-bandwidth 1000
network 172.16.0.0 0.0.255.255 area 0.0.0.0
```

- Router ping:

```
interfaces {
  fe-0/0/1 {
    unit 0 {
      description "link to alex";
      family inet {
        address 172.16.1.2/24;
      }
    }
  }
  fe-1/0/0 {
    unit 0 {
```

```

        description "link to pong";
        family inet {
            address 172.16.2.2/24;
        }
    }
}

routing-options {
    router-id 172.16.2.2;
}

protocols {
    ospf {
        area 0.0.0.0 {
            interface fe-0/0/1.0;
            interface fe-1/0/0.0;
        }
    }
}

```

- Router pong:

```

interfaces {
    fe-0/0/1 {
        unit 0 {
            description "link to alex";
            family inet {
                address 172.16.4.2/24;
            }
        }
    }
    fe-1/0/0 {
        unit 0 {
            description "link to ping";
            family inet {
                address 172.16.3.1/24;
            }
        }
    }
    fe-1/0/1 {
        unit 0 {
            description "link to rolf";
            family inet {
                address 172.16.3.1/24;
            }
        }
    }
}

routing-options {
    router-id 172.16.2.1;
}

protocols {
    ospf {
        area 0.0.0.0 {
            interface fe-0/0/1.0;
            interface fe-1/0/0.0;
            interface fe-1/0/1.0;
        }
    }
}

```

References

- [1] J. Moy. OSPF Version 2. RFC 2328. Available at <http://www.ietf.org/rfc/rfc2328>, 1998. Last accessed May 2008.
- [2] Juniper Networks. NETCONF API Guide. Release 9.1. Available at <http://www.juniper.net/techpubs/software/junos/junos91/netconf-guide/netconf-guide.pdf>, 2008. Last accessed May 2008.
- [3] Cisco Systems. Network Configuration Protocol. Available at http://www.cisco.com/en/US/docs/ios/netmgmt/configuration/guide/nm_cns_netconf.pdf, 2008. Last accessed May 2008.

Problem Y.2: YANG leafs, leaf-lists, lists, container

(10 minutes)

Consider the following four YANG fragments (1),..., (4):

```
container a;      leaf a {          leaf-list a {      list a {
                  type empty;      type string;      key b;
                  }                }                leaf b {
                                }                type empty;
                                }                }
                                }
```

Explain against which of the four YANG fragments the following XML fragments can be validated.

- a) <a/>
 - b) <a>
 - c) <a><a/>
 - d) <a>b
 - e) <a>b<a>c
 - f) <a>
-

Problem Y.3: YANG groupings and choices

(10 minutes)

Consider the following YANG module:

```
module foo {  
  
    namespace "urn:foo";  
    prefix foo;  
  
    grouping x {  
        choice d {  
            case e {  
                leaf f { type empty; }  
            }  
            case g {  
                leaf h { type string; }  
            }  
        }  
    }  
  
    container a {  
        list b {  
            key c;  
            leaf c { type uint8; }  
            uses x;  
            // uses x { choice d { mandatory true; } }  
        }  
    }  
}
```

Which of the following XML documents are valid?

- a) <c>42</c><f/>
- b) <c>42</c><h>a</h>
- c) <c>42</c><f>a</f>
- d) <c>424</c><f></f>
- e) <c>42</c><e><f/></e>
- f) <c>42</c><d><f/></d>
- g) <c>42</c><d><e><f/></e></d>
- h) <c>42</c>

How do your answers change if the line `uses x;` is replaced with the following line?

```
uses x { choice d { mandatory true; } }
```

Problem Y.4: YANG module for VLANs

(20 minutes)

Consider the following XML document representing the configuration of VLANs:

```
<bridge>
  <vlan-capacity>128</vlan-capacity>
  <vlans>
    <vlan>
      <id>1234</id>
      <name>engineering</name>
      <egress-ports>
        <port>2</port>
        <port>4</port>
      </egress-ports>
      <untagged-ports>
        <port>2</port>
      </untagged-ports>
      <status>active</status>
    </vlan>
    <vlan>
      <id>4321</id>
      <name>research</name>
      <egress-ports>
        <port>5</port>
        <port>7</port>
      </egress-ports>
      <untagged-ports>
        <port>7</port>
      </untagged-ports>
      <status>busy</status>
    </vlan>
  </vlans>
</bridge>
```

- a) Write a YANG module representing a possible data model for the VLAN configuration. Try to use reusable constructs.
 - b) Check your YANG module using `pyang`, fixing errors as needed.
 - c) Convert the YANG module to XML schema using `pyang`.
 - d) Use `xmllint` to validate the instance document against the XML schema.
-

Problem Y.5: *VLAN statistics module augmenting the VLAN module*

(15 minutes)

Write a second YANG module which extends the VLAN list by providing basic statistics such as the count of octets transmitted / received per VLAN. Check the module using `pyang`.

References

- [1] M. Bjorklund. YANG - A data modeling language for NETCONF. Internet Draft (work in progress) <draft-ietf-netmod-yang-00.txt>, Tail-f Systems, May 2008.
- [2] J. Schönwälder. Common YANG Data Types. Internet Draft (work in progress) <draft-schoenw-netmod-yang-types-00.txt>, Jacobs University, May 2008.
- [3] Yang Design Team. Yang Central Web Site. Available at <http://www.yang-central.org/>, 2008. Last accessed May 2008.