Automating Network and Service Configuration Using NETCONF and YANG

Stefan Wallin
Luleå University of Technology
stefan.wallin@ltu.se

Claes Wikström
Tail-f Systems AB
klacke@tail-f.com

Abstract

Network providers are challenged by new requirements for fast and error-free service turn-up. Existing approaches to configuration management such as CLI scripting, device-specific adapters, and entrenched commercial tools are an impediment to meeting these new requirements. Up until recently, there has been no standard way of configuring network devices other then SNMP and SNMP is not optimal for configuration management. The IETF has released NETCONF and YANG which are standards focusing on Configuration management. We have validated that NETCONF and YANG greatly simplify the configuration management of devices and services and still provide good performance. Our performance tests are run in a cloud managing 2000 devices.

Our work can help existing vendors and service providers to validate a standardized way to build configuration management solutions.

1 Introduction

The industry is rapidly moving towards a service-oriented approach to network management where complex services are supported by many different systems. Service operators are starting a transition from managing pieces of equipment towards a situation where an operator is actively managing the various aspects of services. Configuration of the services and the affected equipment is among the largest cost-drivers in provider networks [9]. Delivering valued-added services, like MPLS VPNS, Metro Ethernet, and IP TV is critical to the profitability and growth of service providers. Time-to-market requirements are critical for new services; any delay in configuring the corresponding tools directly affects deployment and can have a big impact on revenue. In recent years, there has been an increasing interest in finding tools that address the complex problem of deploying service configurations. These tools need to replace the current configuration management practices that are dependent on pervasive manual work or ad hoc scripting. Why do we still apply these sorts of blocking techniques to the configuration management problem? As Enck [9] points out, two of the primary reasons are the variations of services and the constant change of devices. These underlying characteristics block the introduction of automated solutions, since it will take too much time to update the solution to cope with daily changes. We will illustrate that a NETCONF [10] and YANG [4] based solution can overcome these underlying challenges.

Service providers need to be able to dynamically adopt the service configuration solutions according to changes in their service portfolio without defining low level device configuration commands. At the same time, we need to find a way to remove the time and cost involved in the plumbing of device interfaces and data models by automating device integration. We have built and evaluated a management solution based on the IETF NETCONF and YANG standards to address these configuration management challenges. NETCONF is a configuration management protocol with support for transactions and dedicated configuration management operations. YANG is a data modeling language used to model configuration and state data manipulated by NETCONF. NETCONF was pioneered by Juniper which has a good implementation in their devices. See the work by Tran [23] et. al for interoperability tests of NETCONF.

Our solution is characterized by the following key characteristics:

1. *Unified YANG modeling* for both services and devices.

2. One database that *combines device configuration and service configuration*.

3. *Rendering* of northbound and southbound interfaces and database schemas from the service and device model. Northbound are the APIs published to users

of NCS, be it human or programmatic interfaces. Southbound is the integration point of managed devices, for example NETCONF.

4. A *transaction engine* that handles transactions from the service order to the actual device configuration deployment.

5. An *in-memory high-performance database*.

To keep the service and device model synchronized, (item 1 and 2 above), it is crucial to understand how a specific service instance is actually configured on each network device. A common problem is that when you tear down a service you do not know how to clean up the configuration data on a device. It is also a well-known problem that whenever you introduce a new feature or a new network device, a large amount of glue code is needed. We have addressed this again with annotated YANG models rather then adaptor development. So for example, the YANG service model renders a northbound CLI to create services. From a device model in YANG we are actually able to render the required Cisco CLI commands and interpret the response without the need for the traditional Perl and Expect scripting. Currently our solution can integrate without any plumbing.

It is important to address the configuration management problem using a transactional approach. The transaction should cover the whole chain including the individual devices. Finally, in order to manipulate configuration data for a large network and many service instances we need fast response to read and write operations. Traditional SQL and file-based database technologies fall short in this category. We have used an in-memory database journaled to disk in order to address performance and persistence at the same time.

The objectives of this research are to determine whether these new standards can help to eliminate the device integration problem and provide a service configuration solution utilizing automatically integrated devices. We have studied challenges around data-model discovery, interface versioning, synchronization of configuration data, multi-node configuration deployment, transactional models, and service modeling issues. In order to validate the approach we have used simulated scenarios for configuring load balancers, web servers, and web sites services. Throughout the use-cases we also illustrate the possibilities for automated rendering of Command Line interfaces as well as User Interfaces from YANG models.

Our studies show that a NETCONF/YANG based configuration management approach removes unnecessary manual device integration steps and provides a platform for multi-device service configurations. We see that problems around finding correct modules, loading them and creating a management solution can largely be automated. In addition to this, the transaction engine in our solution combined with inherent NETCONF transaction capabilities resolves problems around multi-device configuration deployment.

We have run performance tests with 2000 devices in an Amazon cloud to validate the performance of NETCONF and our solution. Based on these tests we see that the solution scales and NETCONF provides a configuration management protocol with good performance.

2 Introduction to NETCONF and YANG

The work with NETCONF and YANG started as a result of an IAB workshop held in 2002. This is documented in RFC 3535 [18].

> "*The goal of the workshop was to continue the important dialog started between network operators and protocol developers, and to guide the IETFs focus on future work regarding network management*."

The workshop concluded that SNMP is not being used for configuration management. Operators put forth a number of requirements that are important for a standards-based configuration management solution. Some of the requirements were:

1. Distinction between configuration data and data that describes operational state and statistics.

2. The capability for operators to configure the network as a whole rather than individual devices.

3. It must be easy to do consistency checks of configurations.

4. The availability of text processing tools such as diff, and version management tools such as RCS or CVS.

5. The ability to distinguish between the distribution of configurations and the activation of a certain configuration.

NETCONF addresses the requirements above. The design of NETCONF has been influenced by proprietary protocols such as Juniper Networks JUNOScript application programming interface [14].

For a more complete introduction see the Communications Magazine article [19] written by Schönwälder et al.

2.1 NETCONF

The Network Configuration Protocol, NETCONF, is an IETF network management protocol and is published in RFC 4741. NETCONF is being adopted by major network equipment providers and has gained strong industry support. Equipment vendors are starting to support NETCONF on their devices, see the NETCONF presentation by Moberg [16] for a list of public known implementations.

NETCONF provides mechanisms to install, manipulate, and delete the configuration of network devices. Its operations are realized on top of a simple Remote Procedure Call (RPC) layer. The NETCONF protocol uses XML based data encoding for the configuration data as well as the protocol messages. NETCONF is designed to be a replacement for CLI-based programmatic interfaces, such as Perl + Expect over Secure Shell (SSH). NETCONF is usually transported over the SSH protocol, using the "NETCONF" sub-system and in many ways it mimics the native proprietary CLI over SSH interface available in the device. However, it uses structured schema-driven data and provides detailed structured error return information, which the CLI cannot provide.

NETCONF has the concept of logical data-stores such as "writable-running" or "candidate" (Figure 1). Operators need a way to distribute changes to the devices and validate them locally before activating them. This is indicated by the two bottom options in Figure 1 where configuration data can be sent to candidate databases in the devices before they are committed to running in production applications.

All NETCONF devices must allow the configuration data to be locked, edited, saved, and unlocked. In addition, all modifications to the configuration data must be saved in non-volatile storage. An example from RFC 4741 that adds an interface named "Ethernet0/0" to the running configuration, replacing any previous interface with that name is shown in Figure 2.

2.2 YANG

YANG is a data modeling language used to model configuration and state data. The YANG modeling language is a standard defined by the IETF in the NETMOD working group. YANG can be said to be tree-structured rather than object-oriented. Configuration data is structured into a tree and the data can be of complex types such as lists and unions. The definitions are contained in modules and one module can augment the tree in another module. Strong revision rules are defined for modules. Figure 3 shows a simple YANG example. YANG is mapped to a NETCONF XML representation on the wire.

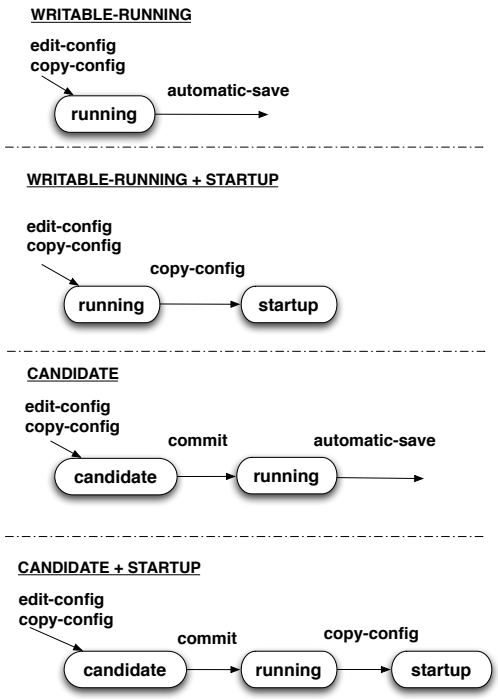

Figure 1: NETCONF Datastores

```
<rpc message-id="101"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <edit-config>
   <target>
      <running/>
   </target>
    <config
    xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
       <top xmlns="http://example.com/schema/1.2/config">
          <interface xc:operation="replace">
            <name>Ethernet0/0</name>
               <mtu>1500</mtu>
               <address>
                 <name>192.0.2.4</name>
                 <prefix-length>24</prefix-length>
              </address>
          </interface>
       </top>
    </config>
 </edit-config>
</rpc>

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 2: NETCONF edit-config Operation

```
module acme-system {
  namespace
  "http://acme.example.com/system";
  prefix "acme";
  organization "ACME Inc.";
  contact "joe@acme.example.com";
  description
    "The ACME system.";
  revision 2007-11-05 {
      description "Initial revision.";
  }
  container system {
    leaf host-name {
        type string;
    }
    leaf-list domain-search {
      type string;
    }
    list interface {
      key "name";
      leaf name {
          type string;
      }
      leaf type {
          type enumeration {
              enum ethernet;
              enum atm;
            }
        }
      leaf mtu {
          type int32;
      }
      must ‘‘ifType != ’ethernet’ or ‘‘+
      ‘‘(ifType = ’ethernet’ and ‘‘ +
      ‘‘mtu = 1500)‘‘ {
      }
...
```

Figure 3: YANG Sample

YANG also differs from previous network management data model languages through its strong support of constraints and data validation rules. The suitability of YANG for data models can be further studied in the work by Xu et. al [24].

3 Our Config Manager Solution - NCS

3.1 Overview

We have built a layered configuration solution, NCS, Network Configuration Server. See Figure 4. The *Device Manager* manages the NETCONF devices in the network and heavily leverage the features of NETCONF and YANG to render a Configuration Manager from the YANG models. At this layer, the YANG models represent the capabilities of the devices and NCS provides the device configuration management capabilities.

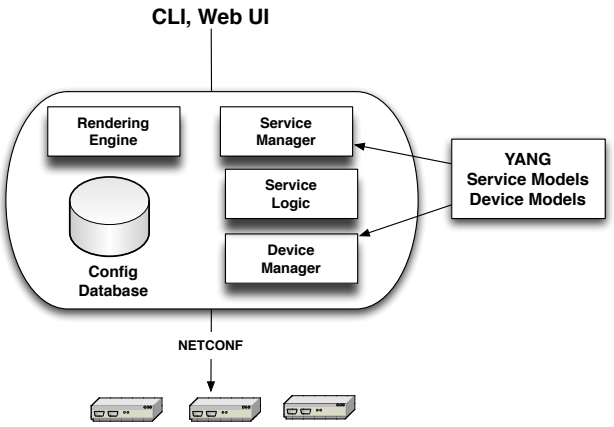

Figure 4: NCS - The Configuration Manager

The *Service Manager* in turn lets developers add YANG service models. For example, it is easy to represent end-to-end connections over L2/L3 devices or web sites utilizing load balancers and web servers. The most important feature of the Service Manager is to transform a service creation request into the corresponding device configurations. This mapping is expressed by defining service logic in Java which basically does a model transformation from the service model to the device models.

The *Configuration Database, (CDB)*, is an in-memory database journaled to disk. CDB is a special-purpose database that targets network management and the in-memory capability enables fast configuration validation and performs diffs between running and candidate databases. Furthermore the database schema is directly rendered from the YANG models which removes the need for mapping between the models and for example a SQL database. A fundamental problem in network management is dealing with different versions of device interfaces. NCS is able to detect the device interfaces through its NETCONF capabilities and this information is used by CDB to tag the database with revision information. Whenever a new model revision is detected, NCS can perform a schema upgrade operation. CDB stores the configuration of services and devices and the relationships between them. NETCONF defines dedicated operations to read the configuration from devices and this drastically reduces the synchronization and reconciliation problem.

Tightly connected to CDB is the *transaction manager*

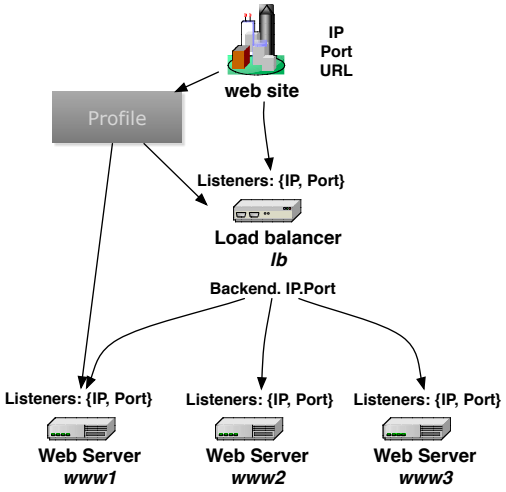

Figure 5: The Example

which manages every configuration change as a transaction. Transactions include all aspects from the service model to all related device changes.

At this point it is important to understand that the NETCONF and NCS approach to configuration management does not use a push and pull approach to versioned configuration files. Rather, it is a fine-grained transactional view based on data models.

The *Rendering Engine* renders the database schemas, a CLI, and a Web UI from the YANG models. In this way the Device Manager features will be available without any coding.

3.2 The Example

Throughout the rest of this paper we will use an example that targets configuration of web-sites across a load balancer and web servers. See Figure 5.

The *service model* covers the aspects of a *web site*; IP Address, Port, and URL. Whenever you provision a web site you refer to a *profile* which controls the selection of load balancers and web servers. A web site allocates a listener on the load balancer which in turn creates backends that refer to physical web servers. So when provisioning a new web site you do not have to deal with the actual load balancer and web server configuration. You just refer to the profile and the service logic will configure the devices. The involved YANG models are :

- `website.yang` : the service model for a web site, it defines web site attributes like url, IP Address, port, and pointer to profile.
- `lb.yang` : the device model for load balancers, it defines listeners and backends where the listeners refers to the web site and backends to the corresponding web servers.
- `webserver.yang` : the device model for a physical web server, it defines listeners, document roots etc.

  The devices in our example are:

  - Load Balancer : `lb`
  - Web Servers : `www1, www2, www3`

3.3 The Device Manager

The Device Manager layer is responsible for configuring devices using their specific data-models and interfaces. The NETCONF standard defines a capability exchange mechanism. This implies that a device reports its supported data-models and their revisions when a connection is established. The capability exchange mechanism also reports if the device supports a `<writable-running>` or `<candidate>` database.

After connection the Device Manager can then use the `get-schema` RPC, as defined in the netconf-monitoring RFC [20] to get the actual YANG models from all the devices. NCS now renders northbound interfaces such as a common CLI and Web UI from the models. The NCS database schema is also rendered from the data-models.

The NCS CLI in Figure 6 shows the discovered capabilities for device "www1". We see that www1 supports 6 YANG data-models, `interfaces`, `webserver`, `notif`, and 3 standard IETF modules. Furthermore the web-server supports NETCONF features like `confirmed-commit, rollback-on-error` and `validation` of configuration data.

In Figure 7 we show a sequence of NCS CLI commands that first uploads the configuration from all devices and then displays the configuration from the NCS configuration database. So with this scenario we show that we could render the database schema from the YANG models and persist the configuration in the configuration manager.

Now, let's do some transaction-based configuration changes. The CLI sequence in Figure 8 starts a transaction that will update the ntp server on www1 and the load-balancer. Note that NCS has the concept of a *candidate* database and a *running*. The first represents the desired configuration change and the running database represents the actual configuration of the devices. At the end of the sequence in Figure 8 we use the CLI command ``compare running brief'' to show the difference between the running and the candidate database. This is what will be committed to the devices. Note that we do a diff and only send the diff. Our in-memory database enables good performance even for large configurations and large networks.

```
ncs> show ncs managed-device www1 capability <RET>
URI                                                    REVISION    MODULE
-------------------------------------------------------------------------------------------------
candidate:1.0                                          -           -
confirmed-commit:1.0                                   -           -
confirmed-commit:1.1                                   -           -
http://acme.com/if                                     2009-12-06  interfaces
http://acme.com/ws                                     2009-12-06  webserver
http://router.com/notif                                -           notif
rollback-on-error:1.0                                  -           -
urn:ietf:params:netconf:capability:notification:1.0    -           -
urn:ietf:params:xml:ns:yang:ietf-inet-types            2010-09-24  ietf-inet-types
urn:ietf:params:xml:ns:yang:ietf-yang-types            2010-09-24  ietf-yang-types
urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring    2010-06-22  ietf-netconf-monitoring
validate:1.0                                           -           -
validate:1.1                                           -           -
writable-running:1.0                                   -           -
xpath:1.0                                              -           -
```

Figure 6: NETCONF Capability Discovery

```
ncs> request ncs sync direction from-device <RET>
...

ncs> show configuration ncs \
managed-device www1 config <RET>

host-settings {
    syslog {
        server 18.4.5.6 {
            enabled;
            selector 1;
        }
...
}
ncs> show configuration ncs \
managed-device lb config  <RET>

lbConfig {
    system {
        ntp-server 18.4.5.6;
        resolver {
            search     acme.com;
            nameserver 18.4.5.6;
        }
    }
}
```

Figure 7: Synchronize Configuration Data from Devices

```
ncs% set ncs managed-device \
www1 config host-settings ntp server 18.4.5.7 <RET>

ncs% set ncs managed-device \
lb config lbConfig system ntp-server 18.4.5.7 <RET>

ncs% compare running brief <RET>

 ncs {
     managed-device lb {
         config {
             lbConfig {
                 system {
-                    ntp-server 18.4.5.6;
+                    ntp-server 18.4.5.7;
                 }
...

ncs% commit
```

Figure 8: Configuring two Devices in one Transaction

In the configuration scenarios shown in Figure 8 we used the auto-rendered CLI based on the native YANG modules that we discovered from the devices. So it gives the administrator one CLI with transactions across the devices, but still with different commands for different vendors in case of non-standard modules. NCS allows for device abstractions, where you can provide a generic YANG module across vendor-specific ones.

Every commit in the scenarios described above resulted in a transaction across the involved devices. In this case the devices support the confirmed-commit capability. This means that the manager performs a commit to the device with a time-out. If the device does not get the confirming commit within the time-out period it reverts to the previous configuration. This is also true for restarts or if the SSH connection closes.

3.4 The Service Manager

In our example we have defined a service model corresponding to web-sites and the corresponding service logic that maps the service model to load balancers and web servers. The auto-rendered Web UI let operators create a web site like the one illustrated in Figure 9.

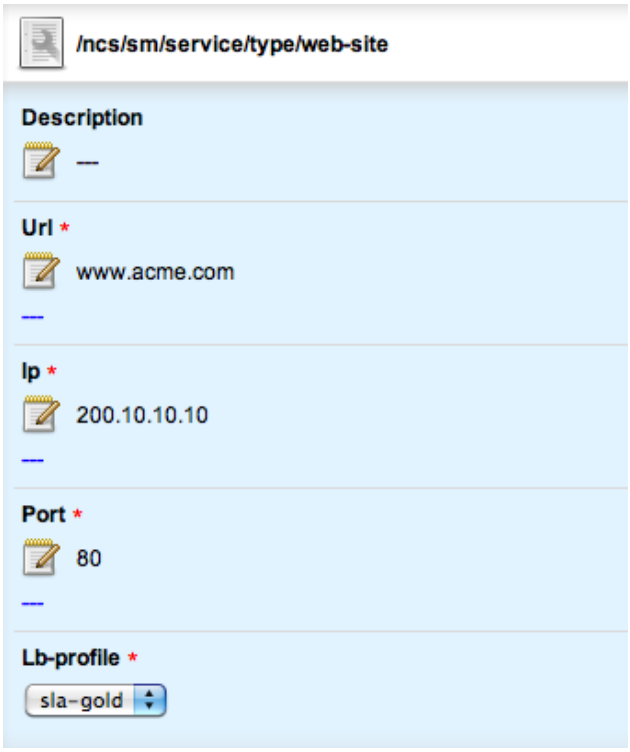

Figure 9: Instantiating a Web-site Service

A fundamental part of the Service Manager is that we use YANG to model services as well as devices. In this way we can ensure that the service model is consistent with the device model. We do this at compile time by checking the YANG service model references to the device model elements. At run-time, the service model constraints can validate elements in the device-model including referential integrity of any references. Let's illustrate this with a simple example. Figure 10 shows a type-safe reference from the web-site service model to the devices. The YANG `leafref` construct refers to a path in the model. The path is verified to be correct according to the model at compile time. At run-time, if someone tries to delete a managed device that is referred to by a service this would violate referential integrity and NCS would reject the operation.

This service provisioning request initiates a hierarchical transaction where the service instance is a parent transaction which fires off child transactions for every

```
leaf lb {
  description "The load balancer to use.";
  mandatory true;
  type leafref {
     path "/ncs:ncs/ncs:managed-device/ncs:name";
   }
}
```

Figure 10: Service-Model Reference to Device-Model

device. In this specific case the selected profile uses all web servers at the device layer. Either the complete transaction succeeds or nothing will happen. As a result the transaction manager stores the resulting device configurations in CDB as shown in Figure 11.

Device-modifications

```
 ncs {
     managed-device lb {
         config {
             lbConfig {
-                listener 200.10.10.10 80 {
-                    service acme-inc {
-                        number 1;
-                        URL-pattern www.acme.com;
-                        backend 192.168.0.9 8008 {
-                        }
-                        backend 192.168.0.10 8008 {
-                        }
-                        backend 192.168.0.11 8008 {
-                        }
-                        session {
-                            type IP;
-                        }
-                    }
-                }
             }
         }
     }
     managed-device www1 {
         config {
             interface eth0 {
-                alias 0 {
-                    ipv4-address 192.168.0.9;
-                }
             }
             wsConfig {
-                listener 192.168.0.9 8008 {
-                }
             }
```

Figure 11: The relationship from a Service to the Actual Device Configurations.

You see that the web-site for `acme` created a listener on the load balancer with backends that maps to the actual web servers. The service also created a listener on the web server. You might wonder why there is a minus-sign for the diff. The reason is that we are actually storing how to delete the service. This means that there will never be any stale configurations in the network. As soon as you delete a service, NCS will automatically clean up.

4 Evaluation

4.1 Performance Evaluation

We have evaluated the performance of the solution using 2000 devices in an Amazon Cloud. The Server is a 4 Core CPU, 4 GB RAM, 1.0 GHz, Ubuntu 10.10 Machine. Here we illustrate 4 test-cases. All test-cases are performed as one single transaction:

1. Start the system with an empty database and upload the configuration over NETCONF from all devices (Figure 12 A).
2. Check if the configuration database is in sync with all the devices (Figure 12 B).
3. Perform a configuration change on all devices (Figure 13 A).
4. Create 500 services instances that touch 2 devices each (Figure 13 B).
5. In Figure 14 we show the memory and database journal disc space for configuring 500 service instances.

All of the test-cases involve the complete transaction including the NETCONF round-trip to the actual devices in the cloud. So, cold-starting NCS and uploading the configuration from 500 devices takes about 8 minutes (Figure 12) and 2000 devices takes about 25 minutes. The configuration synchronization check utilizes a transaction ID to compare the last performed change from NCS to any local changes made to the device. This test assumes that there is some way to get a transaction ID or checksum from the device that corresponds to the last change irrespective of which interface is used. If that is not available and you had to get the complete configuration, then the numbers would be higher.

Updating the config on 500 devices takes roughly one minute, (Figure 8). As seen by Figure 14 the in-memory database has a small footprint even for large networks. In this scenario it is important to note that we always diff the configuration change within NCS before sending it to the device. This means that we only send the actual changes that are needed and this database comparison is included in the numbers. This is an area where we have seen performance bottlenecks in previous solutions when traditional database technologies are used.

These performance tests cover two aspects: performance of NETCONF, and our actual implementation.

NETCONF as a protocol ensures that we achieve at least equal performance to CLI screen scraped solutions and superior performance to SNMP based configuration solutions. XML processing is considerably less CPU intensive than SSH processing.

When running a transaction that touches many managed devices, we use two tricks that affect performance. We pipeline NETCONF RPCs, sending several RPCs in a row, and collecting all the replies in a row. We can also (in parallel) send the requests to all participating managed devices, and then (in parallel) harvest the pipelined replies.

NCS is implemented in Erlang [3, 11] and OTP (Open Telecom Platform) [22] which have excellent support for concurrency and multi-core processors. A lot of effort has gone into parallelizing the southbound requests. For example initial NETCONF SSH connection establishment is done in parallel, greatly enhancing performance.

The network configuration data is kept in a RAM database together with a disk journaling component. If the network is huge, the amount of RAM required can be substantial. When the YANG files are compiled we hash all the symbols in the data models, thus the database is actually a large tree of integers. This increases processing speed and decreases memory footprint of the configuration daemon. The RAM database itself is implemented as an Erlang driver that uses skip lists [17].

Our measurements show that we can handle thousands of devices and hundred thousands of services on off-the-shelf hardware, (4 Core CPU, 4 GB RAM, 1.0 GHz).

We have also made some measurements comparing SNMP and NETCONF performance. We read the interface table using SNMP get-bulk and NETCONF get. In general NETCONF performed 3 times quicker than SNMP. The same kind of performance improvements using NETCONF rather than SNMP can be found in the work by Yu and Ajarmeh [25].

4.2 NETCONF/YANG Evaluation

Let's look at the requirements set forth by RFC 3535 and validate these based on our implementation.

4.2.1 Distinction between configuration data, and data that describes operational state and statistics

This requirement is fulfilled by YANG and NETCONF in that you can explicitly request to get only the configuration data from the device, and elements in YANG are annotated if they are configuration data or not. This greatly simplifies the procedure to read and synchronize configuration data from the devices to a network management system. In our case, NCS can easily synchronize its configuration database with the actual devices.

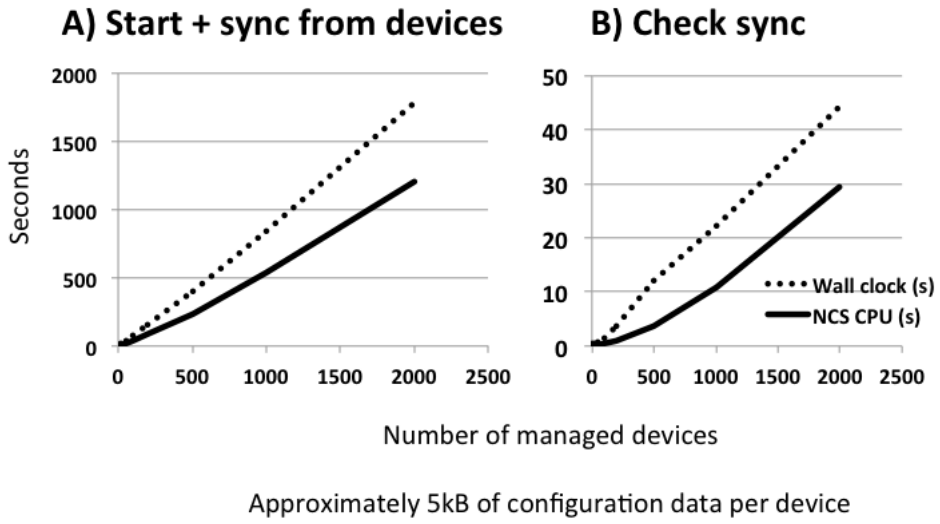

Figure 12: Starting NCS and Reading the Configuration from all Devices (Dotted line represents Wall Clock Time, Solid Line CPU Time).

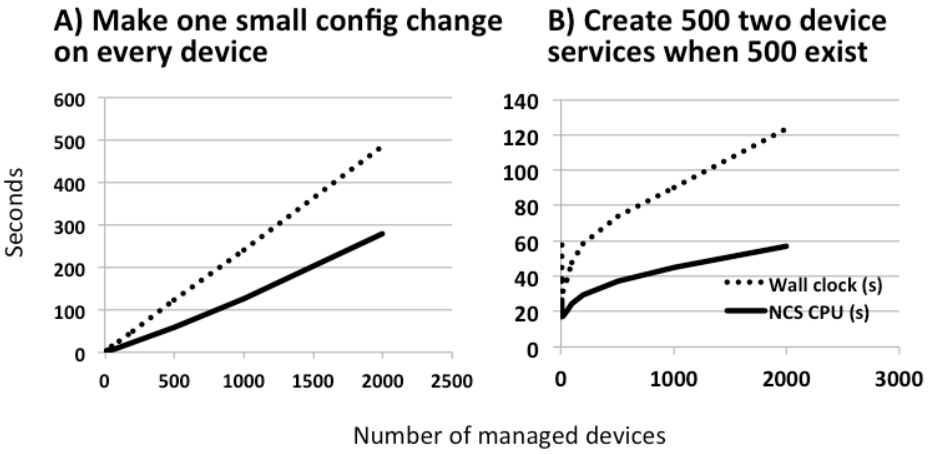

Figure 13: Making Device Configurations and Service Configurations

4.2.2 It is necessary to enable operators to concentrate on the configuration of the network as a whole rather than individual devices

We have validated this from two perspectives

1. Configuring a set of devices as one transaction.

2. Transforming a service configuration to the corresponding device configurations.

Using NCS, we can apply configurations to a group of devices and the transactional capabilities of NETCONF will make sure that the whole transaction is applied or no changes are made at all. The NETCONF `confirmed-commit` operation has proven to be especially useful in order to resolve failure scenarios. A problem scenario in network configuration is that devices may become unreachable after a reconfiguration. The `confirmed-commit` operation requests the device to take the new configuration live but if an acknowledge-

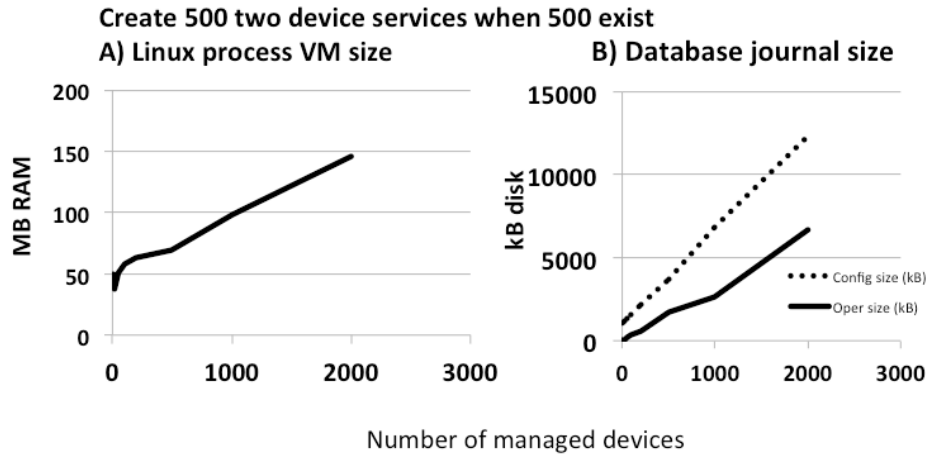

Figure 14: Memory and Journaling Disc Space

ment is not received within a time-out the device automatically rolls-back. This is the way NCS manages to roll-back configurations if one or several of the devices in a transaction does not accept the configuration. It is notable to see the lack of complex state-machines in NCS to do roll-backs and avoid multiple failure scenarios.

In some cases, you would like to apply a global configuration change to all your devices in the network. In the general case the transaction would fail if one of the devices was not reachable. There is an option in NCS to backlog unresponsive devices. In this case NCS will make the transaction succeed and store outstanding requests for later execution.

4.2.3 It must be easy to do consistency checks of configurations.

Models in YANG contain ‘‘`must`’’ expressions that put constraints on the configuration data. See for example Figure 3 where the must expression makes sure that the MTU is set to correct size. So for example, a NETCONF manager can edit the candidate configuration in a device and ask the device to validate it. In NCS we also use YANG to specify service models. In this way we can use must expressions to make sure that a service configuration is consistent including the participating devices. Figure 15 shows a service configuration expression that verifies that the subnet only exists once in the VPN.

4.2.4 It is highly desirable that text processing tools [...] can be used to process configurations.

Since NETCONF operations use well-defined XML payloads, it is easy to process configurations. For example

```
must  "count(
    ../../mv:access-link[subnet =
    current()/../subnet]) = 1" {
      error-message "Subnet must be unique
      within the VPN";
    }
```

Figure 15: Service Configuration Consistency

doing a diff between the configuration in the device versus the desired configuration in the management system. The CLI output in Figure 16 shows a diff between a device configuration and the NCS Configuration Database. In this case a system administrator has used local tools on web server 1 and changed the document root, and removed the listener.

4.2.5 It is important to distinguish between the distribution of configurations and the activation of a particular configuration.

The concept of multiple data-stores in NETCONF lets managers push the configuration to a candidate database, validate it, and then activate the configuration by committing it to the running datastore. Figure 17 shows an extract from the NCS trace when activating a new configuration in web server 2.

```
ncs> request ncs managed-device \
www1 compare-config outformat cli <RET>

diff
 ncs {
     managed-device www1 {
         config {
             wsConfig {
                 global {
-                    ServerRoot /etc/doc;
+                    ServerRoot /etc/docroot;
                 }
-                listener 192.168.0.9 8008 {
-                }
             }
         }
     }
 }
```

Figure 16: Comparing Configurations

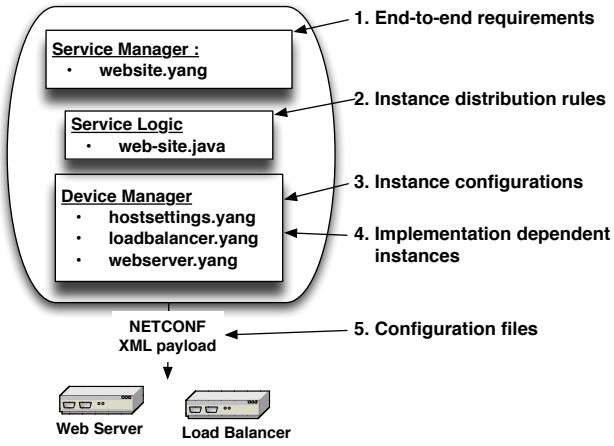

Figure 18: NCS in the Configuration Taxonomy Defined by Delaet and Joosen

```
ncs% set ncs managed-device \
www2 config wsConfig global ServerRoot /etc/doc <RET>

ncs% commit | details <RET>
  ncs: SSH Connecting to admin@www2
  ncs: Device: www2 Sending edit-config
  ncs: Device: www2 Send commit
Commit complete.
```

Figure 17: Separation of Distribution of Configurations and Activation

5 Related Work

5.1 Mapping to Taxonomy of Configuration Management Tools

We can map our solution to other Configuration Management solutions based on the taxonomy defined by Delaet and Joosen [7]. They define a taxonomy based on 4 criteria: abstraction level, specification language, consistency, and distributed management.

The abstraction level ranges from high-level end-to-end requirements to low-level bit-requirements. As shown in Figure 18 and described below, in our solution we work with level 1-5 of the 6 mentioned abstraction levels.

1. *End-to-end Requirements* - The service models in the Service Manager expresses end-to-end requirements including constraints expressed as XPATH must expressions. In the case of our web site provisioning example this corresponds to the model for a web site - `website.yang`.

2. *Instance Distribution Rules* - How an end-to-end service is allocated to resources is expressed in the Java Service Logic Layer. In this layer we map the provisioning of a web site to the corresponding load balancer and web-server models.

3. Instance Configurations - The changed configuration of devices in the Device Manager. The result of the previous point is a diff, configuration change, sent to NCS Device Manager. The Device Manager has two layers. The device independent layer that can abstract different data-models for the same feature and the concrete device model layer. This layer may be vendor-independent. In Figure 18 we indicate a vendor-independent `hostsetting.yang` model which contains a unified model for host settings like DNS and NTP.

4. *Implementation Dependent Instances* - The concrete device configuration in the NCS Device Manager. This is the actual configuration that is sent to the devices in order to achieve the service instantiation. In the specific example of a web site it is the configuration change to the load balancers and web servers.

5. *Configuration Files* - The NETCONF XML, `editconfig`, payload sent to the devices. Note however whereas most tools work with configuration files, NETCONF does fine-grained configuration changes.

6. *Bit-Configurations* - Disk images are not directly managed by NETCONF as such.

When it comes to the specification language we have a uniform approach based on YANG at all lev-

els. Delaet and Joosen characterize the specification language from four perspectives: language-based or user-interface-based, domain coverage, grouping mechanism and multi-level specification. We will elaborate on these perspectives below.

We certainly focus on a *language-based approach* which can render various interface representations. Users can edit the configuration using the auto-rendered CLI and Web UI. You can also feed NCS with the NETCONF XML encoding of the YANG models. NCS is a *general purpose* solution in that the domain is defined by the YANG models and not the system itself. YANG supports *groupings* at the modeling level and NCS supports groupings of instance configurations as configurable templates. Templates can be applied to groups of devices.

NCS supports multi-level specifications which in Delaets and Joosens taxonomy refers to the ability to transform the configuration specifications to other formats. In our case, we are actually able to render Cisco CLI commands automatically from the configuration change. This is a topic of its own and will not be fully covered here. However NCS supports YANG model-driven CLI engines that can be fed with a YANG data-model and the engine is capable of rendering the corresponding CLI commands.

Consistency has three perspectives in the taxonomy: dependency modeling, conflict management, and workflow management. We do not cover workflow management. We consider workflow systems to be a client to NCS. NCS manages dependencies and conflicts based on constraints in the models and runtime policies. The model constraints specifiy dependencies and rules that are constrained by the model itself while policies are runtime constrained defined by system administrators. We use XPATH [6] expressions in both contexts.

Regarding conflict management NCS will detect conflicts as violations to policies as described above. The result is an error message when the user tries to commit the conflicting configuration.

The final component of the taxonomy covers the aspect of *distribution*. NCS supports a fine-grained AAA system that lets different users and client systems perform different tasks. The agent is a NETCONF client on the managed devices. The NCS server itself is centralized. The primary reason here is to enable quick validation of cross-device policy validation. The performance is guaranteed by the in-memory database.

5.2 Comparison to other major configuration management tools

There are many well-designed configuration management tools like: CFengine [5], Puppet [15], LCFG [2] and Bcfg2 [8]. These tools are more focused on system and host configuration whereas we focus mostly on network devices and network services. This is mostly determined by the overall approach taken for configuration management. In our model the management system has a data-model that represents the device and service configuration. Administrators and client programs express an imperative desired change based on the data-model. NCS manages the overall transaction by the concept of a candidate and running database which is a well-established principle for network devices.

Many host-management uses concepts of centralized versioned configuration files rather than a database with roll-back files. Also in a host environment you can put your specific agents on the hosts which is not the case for network devices. Therefore a protocol based approach like NETCONF/YANG is needed.

Another difference is the concept of desired state. For host configuration it is important to make sure that the hosts follow a centrally defined configuration which is fairly long-lived. In our case we are more focused on doing fine-grained real-time changes based on requirements for new services. There is room for combination of the two approaches where host-based approaches focused on configuration files address the more static setup of the device and our approach on top if that addresses dynamic changes.

It is also worth-while noting that most of the existing tools have made up their own specific languages to describe configuration. YANG is a viable options for the above mentioned tools to change to a standardized language.

There is of course a whole range of commercial tools, like Telcordia Activator [21], HP Service Activator [12], Amdocs [1], that address network and service configuration. While they are successfully being used for service configuration, the underlying challenges of cost and release-cycles for device adapters and flexibility of service models can be a challenge.

6 Conclusion and Future Work

6.1 Conclusion

We have shown that a standards-based approach for network configuration based on NETCONF and YANG can ease the configuration management scenarios for operators. Also the richness of YANG as a configuration description language lends itself to automating not only the device communication but also the rendering of interfaces like Command Line Interfaces and Web User Interfaces. Much of the value in this IETF standard lies in the transaction-based approach to configuration management and a rich domain-specific language to describe the

configuration and operational data. We used Erlang and in-memory database technology for our reference implementation. These two choices provide performance for parallel configuration requests and fast validation of configuration constraints.

6.2 Future Work

We have started to work on a NETCONF SNMP adaptation solution which is critical to migrate from current implementations. This will allow for two scenarios: read-only and read-write. The read-only view is a direct mapping of SNMP MIBs to corresponding NETCONF/YANG view, this mapping is being standardized by IETF [13]. The read-write view is more complex and cannot be fully automated. The main reason is that the transactional capabilities and dependencies between MIB variables are not formally defined in the SNMP SMI, for example it is common that you need to set one variable before changing others. We are working on catching the most common scenarios and define YANG extensions for those in order to automatically render as much as possible.

Furthermore we are working on a solution where we can have hierarchical NCS systems in order to cover huge networks like nation-wide Radio Access Networks. We will base this on partitioning of the instantiated model into separate CDBs. NCS will then proxy any NETCONF requests to the corresponding NCS system.

We are also working on two interesting features in order to understand the service configuration versus the device configuration: "dry-run" and "service check-sync". Committing a service activation request with dry-run calculates the resulting configuration changes to the devices and displays the diff without committing it. This is helpful in a what-if scenario: "If I provision this VPN, what happens to my devices?". The service check-sync feature will compare a service instance with the actual configuration that is on devices and display any conflicting configurations. This is useful to detect and analyze if and how the device configurations have been changed by any local tools in a way that breaks the service configurations.

References

[1] AMDOCS. Amdocs service fulfillment, 2011. http://www.amdocs.com/Products/OSS/Pages/Service-Fulfillment.aspx.

[2] ANDERSON, P., SCOBIE, A., ET AL. Lcfg: The next generation. In *UKUUG Winter conference* (2002), Citeseer.

[3] ARMSTRONG, J., VIRDING, R., WIKSTRÖM, C., AND WILLIAMS, M. Concurrent Programming in ERLANG, 1993.

[4] BJORKLUND, M. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020 (Proposed Standard), Oct. 2010.

[5] BURGESS, M. A tiny overview of cfengine: Convergent maintenance agent. In *Proceedings of the 1st International Workshop on Multi-Agent and Robotic Systems, MARS/ICINCO* (2005), Citeseer.

[6] CLARK, J., DEROSE, S., ET AL. XML path language (XPath) version 1.0. *W3C recommendation* (1999).

[7] DELAET, T., AND JOOSEN, W. Survey of configuration management tools. *Katholieke Universiteit Leuven, Tech. Rep* (2007).

[8] DESAI, N., LUSK, A., BRADSHAW, R., AND EVARD, R. Bcfg: A configuration management tool for heterogeneous environments. *Cluster Computing, IEEE International Conference on 0* (2003), 500.

[9] ENCK, W., MCDANIEL, P., SEN, S., SEBOS, P., SPOEREL, S., GREENBERG, A., RAO, S., AND AIELLO, W. Configuration management at massive scale: System design and experience. In *Proc. of the 2007 USENIX: 21st Large Installation System Administration Conference (LISA '07)* (2007), pp. 73–86.

[10] ENNS, R. NETCONF Configuration Protocol. RFC 4741 (Proposed Standard), Dec. 2006.

[11] ERLANG.ORG. The erlang programming langugage, 2011. http://www.erlang.org/.

[12] HP. HP Service Activator, 2011. http://h20208.www2.hp.com/cms/solutions/ngoss/fulfillment/hpsa-suite/index.html.

[13] J. SCHÖNWÄLDER. Translation of SMIv2 MIB Modules to YANG Modules. Internet-Draft, July 2011. http://tools.ietf.org/html/draft-ietf-netmod-smi-yang-01.

[14] JUNIPER. Junos XML Management Protocol, 2011. http://www.juniper.net/support/products/junoscript/.

[15] KANIES, L. Puppet: Next-generation configuration management.; login: the USENIX Association newsletter, 31 (1), 2006.

[16] MOBERG, C. A 30 Minute Introduction To NETCONF and YANG, 2011. http://www.slideshare.net/cmoberg/a-30minute-introduction-to-netconf-and-yang.

[17] PUGH, W. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM 33* (June 1990), 668–676.

[18] SCHOENWAELDER, J. Overview of the 2002 IAB Network Management Workshop. RFC 3535 (Informational), May 2003.

[19] SCHÖNWÄLDER, J., BJÖRKLUND, M., AND SHAFER, P. Network configuration management using NETCONF and YANG. *Communications Magazine, IEEE 48*, 9 (sept. 2010), 166 –173.

[20] SCOTT, M., AND BJORKLUND, M. YANG Module for NETCONF Monitoring. RFC 6022 (Proposed Standard), Oct. 2010.

[21] TELCORDIA. Telcordia activator, 2011. http://www.telcordia.com/products/activator/index.html.

[22] TORSTENDAHL, S. Open telecom platform. *Ericsson Review(English Edition) 74*, 1 (1997), 14–23.

[23] TRAN, H., TUMAR, I., AND SCHÖNWÄLDER, J. Netconf interoperability testing. In *Scalability of Networks and Services*, R. Sadre and A. Pras, Eds., vol. 5637 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009, pp. 83–94. 10.1007/978-3-642-02627-0-7.

[24] XU, H., AND XIAO, D. Considerations on NETCONF-Based Data Modeling. In *Proceedings of the 11th Asia-Pacific Symposium on Network Operations and Management: Challenges for Next Generation Network Operations and Service Management* (2008), Springer, p. 176.

[25] YU, J., AND AL AJARMEH, I. An Empirical Study of the NETCONF Protocol. In *Networking and Services (ICNS), 2010 Sixth International Conference on* (march 2010), pp. 253 –258.