Towards Validated Network Configurations with NCGuard

Laurent VANBEVER, Grégory PARDOEN, Olivier BONAVENTURE

INL: IP Networking Lab (http://inl.info.ucl.ac.be, laurent.vanbever@uclouvain.be)

Université catholique de Louvain (UCL), Belgium

Internet Network Management Workshop
October 19, 2008
Agenda

• Introduction
  • State-of-the art in network configuration

• NCGuard: Towards new configuration paradigm
  • High-level representation
  • Validation
  • Generation

• Conclusion

• Demo session (1:30pm - 2:30pm)
Introduction
Some networking facts

- Configuring networks is **complex, costly**, and **error-prone**
- Networks can be composed of hundreds to thousands of devices
  - **Manual** configuration, **equipment-by-equipment**
  - **Trial-and-error** approach
- **Diversity** of vendor-specific languages (IOS, JunOS, etc.)
  - Syntax, semantic, and supported features sets are different
- **Low-level** configuration languages
  - Lot of code duplication
Consequences

- Network misconfigurations are **frequent**

  - “Human factors, is the biggest contributor — responsible for **50 to 80 percent** of network device outages”\(^1\)

  - In 2002, **0.2% to 1%** of the BGP table size suffer from misconfiguration\(^2\)

- Misconfigurations have led and **still lead** to large scale problems (e.g., YouTube in 2008)

- Management costs **keep growing** due to the increasing complexity of network architectures

---

Current Approaches: Static Analysis

• **Use pattern matching** on configurations to detect misconfigurations \(^1\)

• Compare configurations to given **specifications** \(^2\)

• **Pro & Con:**
  - Very effective to detect some critical problems
  - Need a *a priori* specifications of what a valid network is
  - Difficulties encountered when analyzing heterogenous networks
  - Solution: use of an intermediate representation

---


Current Approaches: Data mining

- Perform **statistical** analysis directly on configurations \(^1\)
- **Infer** network-specific policies, then perform **deviation** analysis \(^2\)

**Pro & Con:**

- Completely independent of *a priori* validity specifications
- Too verbose, people are flooded with non-error messages.
- Difficulties encountered when analyzing heterogenous networks
  - Solution: use of an intermediate representation


Current Approaches: Design

 Legend:

- **INPUT**
- **PROCESS**
- **OUTPUT**

**Specifications** → **Validator** → **Intermediate Representation** → **Translator** → **Device 1 Config.** → **Device 2 Config.** → **Device N Config.** → **Validated!** → **Errors & Warnings**

**Bottom-Up Approach**
NCGuard: Towards new configuration paradigm\textsuperscript{1}

\textsuperscript{1} \url{http://inl.info.ucl.ac.be/softwares/ncguard-network-configuration-safeguard}
Starting point

• Network configuration contrasts with numerous progress in **software engineering**
  • Requirements, specifications, verification, validation, new development schemes, etc.
  • In comparison, network configuration is like *writing a distributed program in assembly language*  

• Current approaches do **not solve** the problem
  • Do not relax the burden associated to the configuration phase
  • **Why not apply** software engineering techniques to network configurations?

---

NCGuard Design

Legend:  
- **INPUT**  
- **PROCESS**  
- **OUTPUT**  

Diagram:

- **SPECIFICATIONS**  
- **NETWORK REPRESENTATION**  
- **VALIDATOR**  
- **ERRORS & WARNINGS**  
- **JUNIPER TEMPLATE**  
- **CISCO TEMPLATE**  
- **DEVICE 1 CONFIG.**  
- **DEVICE 2 CONFIG.**  
- **DEVICE N CONFIG.**  

**TOP-DOWN APPROACH**
Main concepts

1. **High-level** representation (i.e., abstraction) of a network configuration
   - Suppress redundancy
   - Vendor-independent

2. Rule-based **validation** engine
   - A rule represents a condition that must be met by the representation
   - Flexible way of adding rules

3. **Generation** engine
   - Produce the configuration of each device in its own configuration language
Validation engine

- After a survey of real network configurations, we found that many rules follow regular patterns.
- In NCGuard, we implemented the structure of several patterns, that can be easily specialized:
  - Presence (or non-presence)
  - Uniqueness
  - Symmetry
- If a rule cannot be expressed as one of them:
  - Custom (e.g., connexity test, network redundancy test, etc.)
Scope: All routers

Rules are expressed formally by using the notions of **scope** and its **descendants**

- A configuration node is an element of the high-level representation
- Composed of fields
- A **scope** is a set of configuration nodes
- **descendants(x)** is a set of selected descendants of the scope’s element \( x \)

### Diagram

- **Routers**
  - **R1**
    - Interface so-0/0/1
    - Interface loopback
  - **R2**
    - Interface so-0/0/1
    - Interface loopback

- **descendants(R1)**: all R1’s interfaces
- **descendants(R2)**: all R2’s interfaces

- : Configuration node
Presence rule

- Check if certain configuration nodes are in the representation

Example: each router *must* have a loopback interface
Presence rule

Check if there is at least one configuration node respecting a given condition in each descendants set.

\[ \forall x \in \text{SCOPE} \ \exists y \in \text{descendants}(x) : C_{\text{presence}}(T, y) \]

Example: each router **must** have a loopback interface

\[ \forall x \in \text{ROUTERS} \ \exists y \in \text{interfaces}(x) : y.id = \text{loopback} \]

```xml
<rule id="LOOPBACK_INTERFACE_ON_EACH_NODE" type="presence">
  <presence>
    <scope>ALL_NODES</scope>
    <descendants>interfaces/interface</descendants>
    <condition>@id='loopback'</condition>
  </presence>
</rule>
```
Uniqueness rule

Check the uniqueness of a field value in a set of configuration nodes

Example: uniqueness of routers interfaces identifiers

Ids of R1’s interfaces are unique.

Ids of R2’s interfaces are not unique. The rule will failed.
Uniqueness rule

Check if there is no two configuration nodes with identical value of field

∀x ∈ SCOPE ∀y ∈ d(x) : ¬(∃z ≠ y ∈ d(x) : y.field = z.field)

Example: uniqueness of routers interfaces identifiers

∀x ∈ ROUTERS ∀y ∈ interfaces(x) : ¬(∃z ≠ y ∈ interfaces(x) : y.id = z.id)

<rule id="UNIQUENESS_INTERFACE_ID" type="uniqueness">
  <uniqueness>
    <scope>ALL_NODES</scope>
    <descendants>interfaces/interface</descendants>
    <field>@id</field>
  </uniqueness>
</rule>
Symmetry rule

- Check the equality of fields of configuration nodes
- Such rules can be checked \textit{implicitly} by the high-level representation
- Example: MTU must be equal on both ends of a link
  - Automatically checked by modeling it once at the link level
  - Instead of twice at the interfaces level
- Hypothesis: duplication phase is correct
Custom rule

- Used to check **advanced** conditions
- Expressed in a query or programming language

Example: All OSPFs areas must be connected to the backbone

```xml
<rule id="ALL_areas_CONNECTED_TO_BACKBONE_AREA" type="custom">
  <custom>
    <xquery>
      <xquery>
        for $area in /domain/ospf/areas/area[@id!="0.0.0.0"]
        let $nodes := $area/nodes/node
        where count(/domain/ospf/areas/area) > 1
        and not(some $y in $nodes satisfies /domain/ospf/areas/
        area[@id="0.0.0.0"]/nodes/node[@id=$y/@id])
      
        return
        <result><area id="{$area/@id}"/></result>
      </xquery>
    </custom>
  </rule>
```
Generation

- High level representation is not designed to be translated into low level language

- **Intermediate** representations are needed

- **Templates** translate those intermediates representations into configuration files

- Support of any configuration or modeling language (e.g., Cisco IOS, Juniper JunOS, etc.)
Generation

```xml
<node id="SALT">
    <interfaces>
        <interface id="lo0">
            <unit number="0">
                <ip type="ipv4" mask="32">198.32.8.200</ip>
                <ip type="ipv6" mask="128">2001:468:16::1</ip>
            </unit>
        </interface>
    </interfaces>
</node>
```

```javascript
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 198.32.8.200/32;
            }
            family inet6 {
                address 2001:468:16::1/128;
            }
        }
    }
}
```
Conclusion
Conclusion

• NCGuard is a **first step** towards an **extensible**, and **easy** way of designing and configuring **correct** networks.

• **Easy** to:
  
  • Add new protocols, equipments, parameters, etc.
  
  • Add rules to check specific needs or new features
  
  • Add new templates to generate appropriate configlets

• **Further works:**
  
  • Extends the prototype to a broader range of case
  
  • Allow NCGuard to interact directly with the routers
Any Questions?