Operational Tools for High Availability

Muku Murthy

APRICOT 2007

Agenda

- Introduction
- The High-Availability Imperative
- Streamlining Network Operations
  - Configuration Policy Enforcement
  - Automatic Configuration Generation
  - Custom Operational Commands
  - Automated Troubleshooting
- Summary

Introduction

- Generic Concepts and Scenarios
- Applicable to both Service Providers and Enterprises
- Examples are specific to Juniper Devices

The High-Availability Imperative

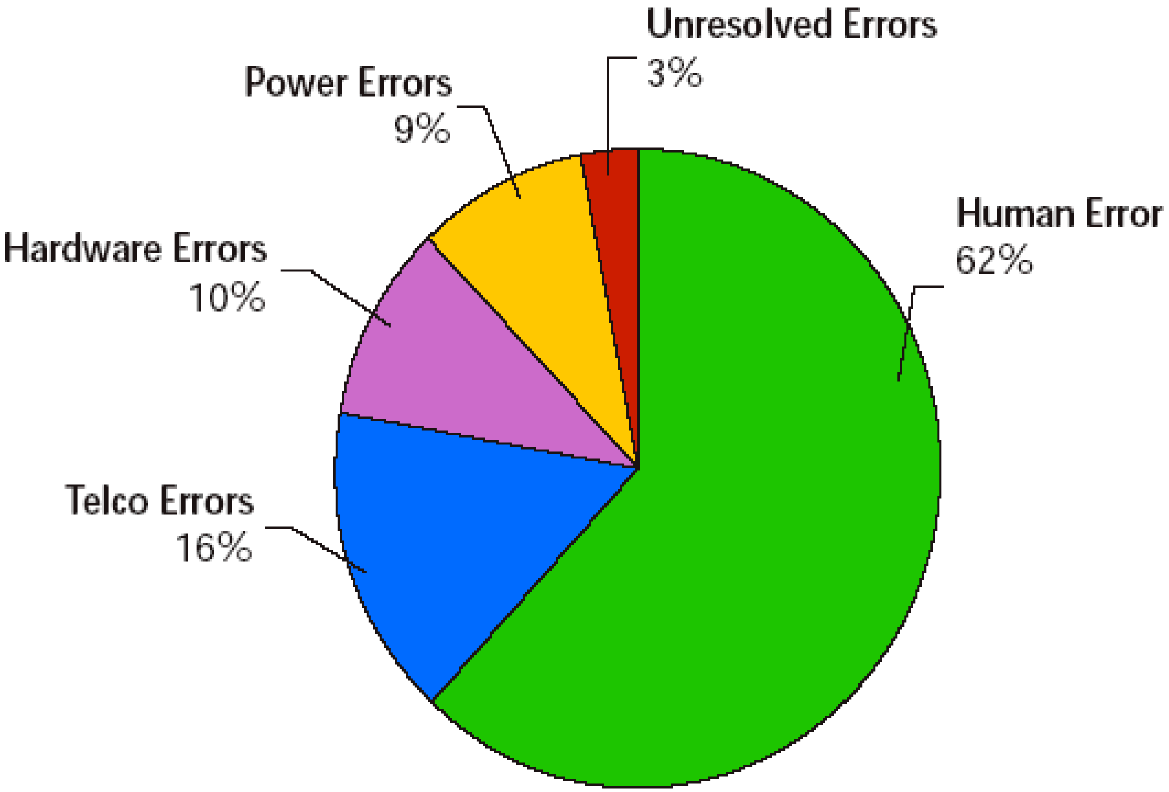
- **Cost of Failure is high and getting higher**
  - Increasing Critical Apps on the Network
    - Video, VoIP, VPNs
    - Web Services, On-Demand Computing
  - Millions of Dollars of Lost Revenue, and Productivity
  - SLAs, Customer expectations

➢ How Available are Networks?

Procedural Errors are the Leading Cause of Network Downtime.

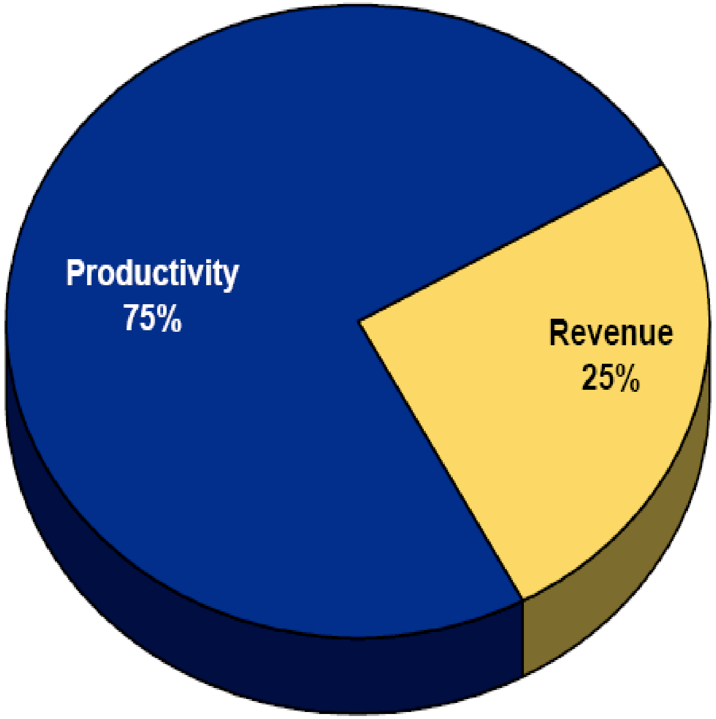
- Here are Some Case Studies...

Case-1: Network Downtime Survey (2002)

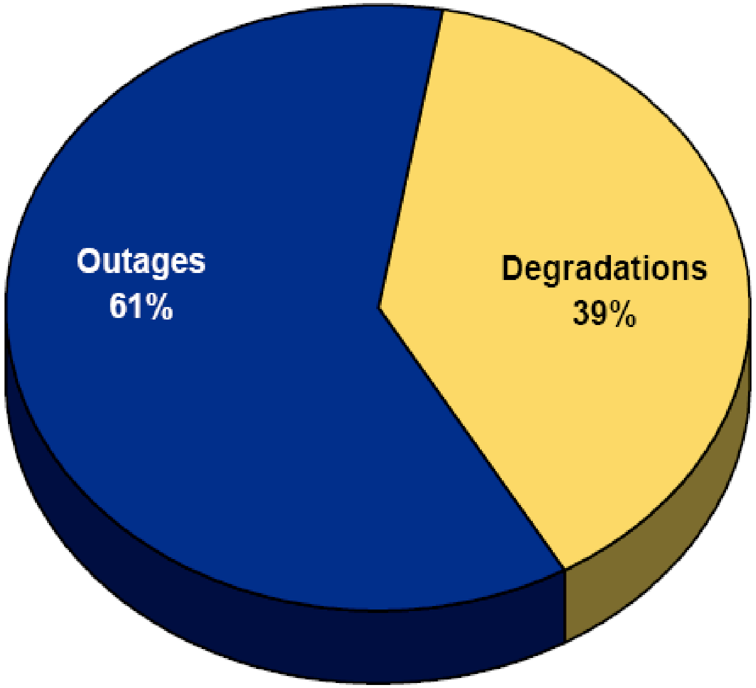


Source: The Yankee Group 2002 Network Downtime Survey

Case-2: Network Downtime Survey (2005)



Annual total: $222M



Annual total: $222M

Source: Infonetics Research 2005 Network Downtime Survey

Case-3: E911 Major Outage Report (2007)

- **Network Reliability Steering Committee Investigated at the Request of FCC**
- **Major E911 Outages are defined as**
  - Affecting 300,000 or more users
  - For 60 minutes or more
- **Outages from January 2005 through July 2006**
  - 73 Major Outage Reports (14% of Total)
  - Nearly Half (49%) of the Major Outages were due to procedural errors

➢ Once procedural errors were identified they were quickly corrected..

What is the Challenge?

Human error is the most troubling, because fixes for human error are elusive and require process changes and retraining, which can take a long time and be very expensive.

Source: Infonetics Research 2005 Network Downtime Survey

- ➢ Solution: Native Device-Level support to enforce the Standard Operating Procedures..

Agenda

- Introduction
- The High-Availability Imperative
- **Streamlining Network Operations**
  - Configuration Policy Enforcement
  - Automatic Configuration Generation
  - Custom Operational Commands
  - Automated Troubleshooting
- Summary

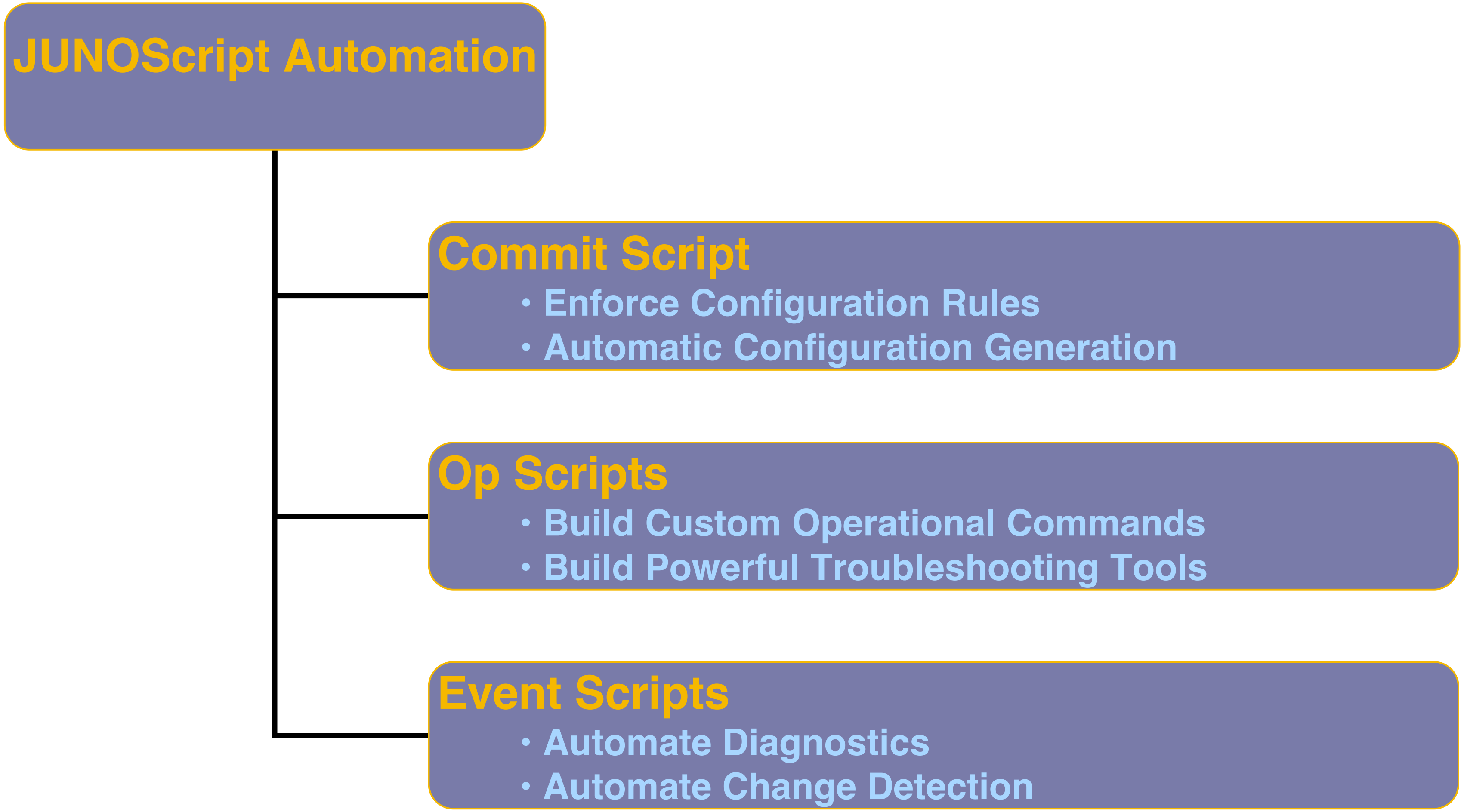
Streamlining Network Operations

- **Build Intelligence and Automation into the Network**
  - Enforce Configuration Validity
  - Simplify Configuration Generation for Complex Services
  - Deploy Powerful Custom Operational Commands
  - Customize/Automate Network Troubleshooting
    - Event-Driven Change Detection
    - Automated Diagnosis
    - Automated Remediation

Imagine Being Able to Enforce…..

- **Valid Service Configuration**
  - All LDP-enabled interfaces are configured for IGP
  - No Accidental deletion of [interfaces] and [protocols] blocks
  - T1 interface configured under [interfaces] has corresponding [protocols rip] configuration
- **Performance Guidelines**
  - Minimum MTU Setting on all SONET interfaces
  - Maximum number of VLANs per port
  - Each ATM interface not to exceed 1000 PVCs
- **Security Guidelines**
  - IKE Authentication Algorithm should be SHA-256
  - All Public exchange peers must have MD5

JUNOScript Automation is the Answer!



JUNOScript Automation

Commit Script

- Enforce Configuration Rules
- Automatic Configuration Generation

Op Scripts

- Build Custom Operational Commands
- Build Powerful Troubleshooting Tools

Event Scripts

- Automate Diagnostics
- Automate Change Detection

JUNOScript Architectural Blocks

- **XML**
- **XPATH**
- **JUNOS Configuration Model**
- **JUNOS XML Output**
  - CLI: `“<operational-command> | display xml”`
  - NETCONF

XML

- eXtensible Markup Language
- Structured, self-describing language
- Individual Elements and their Hierarchical Relationships
- XML documents are easily parsed and can overcome the problem of vendor specific CLI grammars and syntax

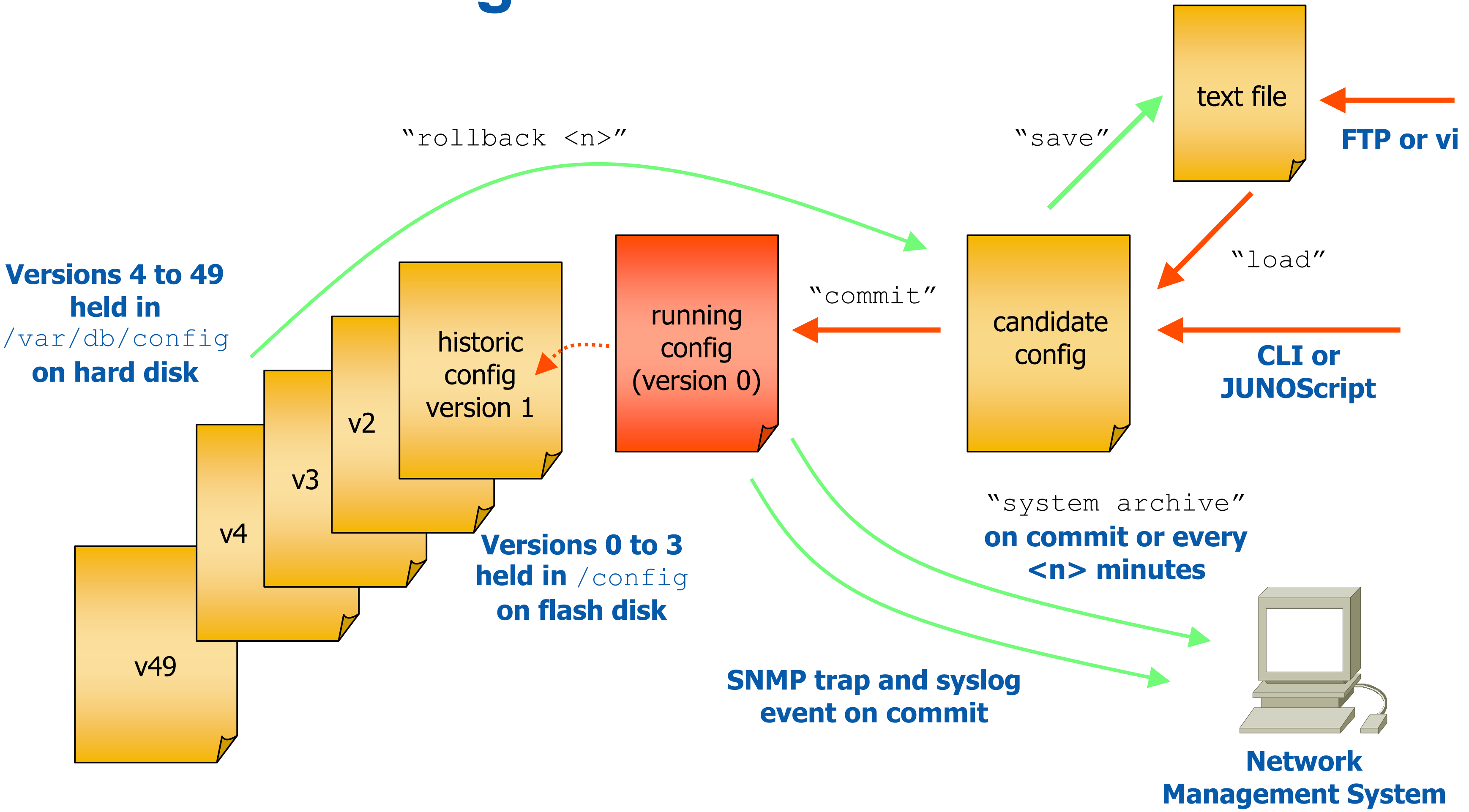
```
<configuration>
  <system>
     <host-name>my-router</host-name>
     <accounting inactive="inactive">
  </system>
</configuration>
```

XPATH

- **Specify and Locate elements in XML hierarchy**
- **Powerful Expression Syntax**
- **Enables Definition of Complex Criteria for Selecting portions of XML hierarchy**
- **Example XPATH Expressions**
  - `/configuration/system/host-name`
  - `*[@inactive]`
  - `host-name[name = '10.1.1.1']`

```
<configuration>
  <system>
     <host-name>my-router</host-name>
     <accounting inactive="inactive">
  </system>
</configuration>
```

JUNOS Configuration Model



Script Deployment Model

- **Design and Develop Scripts with due consideration to**
  - Service Deployment Decisions
  - Standard Operating Procedures
  - Scripting Best Practices
- **Deploy Scripts on Device**
  - NETCONF or file copy to specific locations
  - Update Device Configuration to include Scripts
  - User permission model applies

Scripting Environment

- SLAX (Simpler, Perl like)
- XSLT (W3C standard)
- File-transfer via SCP/FTP
- Extensive ‘debugging’ possible

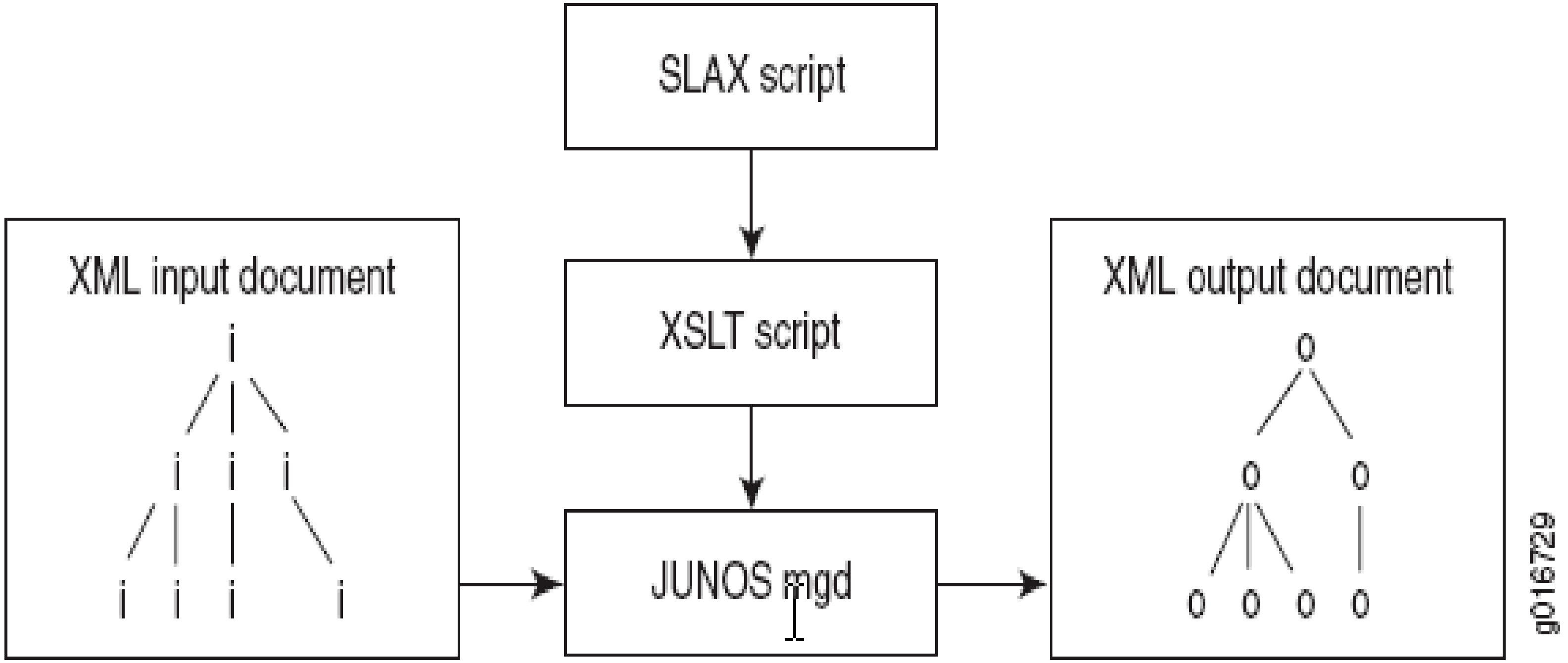
Agenda

- Introduction
- The High-Availability Imperative
- **Streamlining Network Operations**
  - Configuration Policy Enforcement
  - Automatic Configuration Generation
  - Custom Operational Commands
  - Automated Troubleshooting
- Summary

Commit Script

- **Commit Scripts are:**
  - Run at commit time
  - Inspect the incoming configuration
  - Perform actions including
    - Failing the commit (self-defense)
    - Modifying the configuration (self-correcting)
- **Commit scripts can:**
  - Generate custom error/warning/syslog messages
  - Make changes or corrections to the configuration
- **Extended configuration checking**
  - **Your** design rules
  - **Your** implementation details
  - 100% of **Your** design standards
- **Commit Scripts allow customers better control over how their devices are configured**

Commit Script Operation



Generate an Error

- **Prevent a commit from succeeding**
  - Tell the user the exact reason
- **Example**

```
for-each (interfaces/interface[starts-with(name, "so-") && mtu && mtu <
   $min-mtu]) {
       <xnm:error> {
           <message> {
              expr "SONET interfaces must have a minimum mtu of 2048";
                      } } }
```

Generate a Warning

- **Inform user of potential problem**
  - But allow the commit to proceed
- **Example**

```
for-each (interfaces/interface[starts-with(name, "so-") && mtu && mtu <
   $min-mtu]) {
       <xnm:warning> {
           <message> {
             expr "SONET interfaces must have a minimum mtu of 2048";
                       } } }
```

Generate a Syslog Message

- **Simple text message passed to syslog()**
  - Can be forwarded to remote server
    - Using normal syslog abilities
- **Use <syslog> element**
- **Example**

```
<syslog>
  <message>Commit by ‘jon’ outside maintenance window</message>
</syslog>
```

Example: Error Checking

- Check if IGP is configured for all so-* interfaces
- Check if MPLS and ISO families are enabled on all core links
- Check if MPLS configured interfaces are present under protocol mpls (and rsvp, …)

…and optionally, correct the error!

Example: Enforce Configuration Rules

- Maximum number of VLANs per port
- All public exchange peers must have MD5
- Firewall has to have trailing explicit deny
- Filter on lo0, telnet disabled
- Certain set of parameters must always be set

…commit prevented until configuration is in compliance with rules.

Example – SLAX Code

```
param $min-mtu = 2048;

match configuration {
        for-each (interfaces/interface[starts-with(name, 'so-')
               and mtu and mtu < $min-mtu]) {
                      <xnm:error> {
                      call jcs:edit-path();
                      call jcs:statement($dot = mtu);
                      <message> {
                              expr "SONET interfaces must have a
minimum MTU of ";
                              expr $min-mtu;
                              expr ".";
                      }
               }
        }
}
```

Example - Device Configuration

```
system {
        scripts {
                commit {
                        file ex-so-mtu.xsl;
                }
        }
}
interfaces {
        so-1/2/2 {
                mtu 2048;
        }
        so-1/2/3 {
                mtu 576;
        }
}
```

Example – Commit Operation Output

```
user@host# commit
[edit interfaces interface so-1/2/3]
'mtu 576;'
SONET interfaces must have a minimum MTU of 2048.
error: 1 error reported by commit scripts
error: commit script failure
```

Automating Configuration Changes

- **Use <change> element**
- **Regular changes**
  - Just like normal CLI changes
  - Add, Delete, Insert, Rename, Activate, Deactivate, Annotate
- **Transient changes**
  - Does not appear in normal config
  - Allows intelligent configuration groups
    - i.e. apply only if a condition is met
- **And for both..**
  - Full access to all JUNOS configuration
  - Full access to show output, etc

Example - Adding T1 Interfaces to a RIP Group

- **This example adds**
  - Every T1 interface configured at the [edit interfaces] hierarchy level
  - To the [edit protocols rip group test] hierarchy level
- **The changes to the configuration are made silently**

Example – SLAX Code

```
match configuration {
   var $all-t1 = interfaces/interface[starts-with(name, 't1-')];
   if ($all-t1) {
      <change> {
         <protocols> {
            <rip> {
               <group> {
                  <name> "test";
                     for-each ($all-t1) {
                        var $ifname = name _ '.0';
                        <neighbor> {
                        <name> $ifname;
                     }
                  }
               }
            }
         }
      }
   }
}
```

Agenda

- Introduction
- The High-Availability Imperative
- **Streamlining Network Operations**
  - Configuration Policy Enforcement
  - Automatic Configuration Generation
  - Custom Operational Commands
  - Automated Troubleshooting
- Summary

Automating Configuration Generation

- **Use Commit Script Macros**
- **Scripts are written and tested by tier-3 engineers**
- **Scripts are uploaded to routers**
- **Operators invoke macro and enter variables**
- **Macro writes the configuration**
  - Complex configurations can be created from simple entries
- **Result:**
  - Simple operator entries sharply reduce configuration errors
  - Configurations written correctly **every time**
  - Configurations written consistently **every time**

Example JUNOScript Macro: Creating a Complex VPLS Config

```
vpls-100 {
    apply-macro vpls-inst {
        id 100;
        interface ge-0/0/0.10;
        site 2;
        via ASD-2A;
    }
}
```

```
routing-instances {
    vpls-100 {
        /* # Generated by vpls-inst.xsl # */
        instance-type vpls;
        interface ge-0/0/0.10;
        route-distinguisher 192.168.0.92:100;
        vrf-export [ CUST_VIA_ASD-2A CUST-vpls-100 ];
        vrf-target import target:100:100;
        protocols {
            vpls {
                site-range 24;
                mac-table-size
                site cressida
                    site-ident
                }
            }
        }
    }
}
```

```
interfaces {
    ge-0/0/0 {
        unit 10 {
            description vpls-100;
            encapsulation vlan-vpls;
            vlan-id 10;
            input-vlan-map {
                swap;
                vlan-id 100;
            }
            output-vlan-map swap;
        }
    }
}
policy-options {
    policy-statement CUST-vpls-100 {
        then {
            community add CUST-vpls-100;
            accept;
        }
    }
    community CUST-vpls-100 members target:100:10
}
```

- Operator invokes macro, specifies VPLS instance variables
- At commit, macro writes complex VPLS configuration which includes Routing Instance, Interface configuration, and Policy entry
- 100% in compliance with configuration rules

Agenda

- Introduction
- The High-Availability Imperative
- **Streamlining Network Operations**
  - Configuration Policy Enforcement
  - Automatic Configuration Generation
  - Custom Operational Commands
  - Automated Troubleshooting
- Summary

Op Scripts Overview

- **Allow scripts to perform operational tasks**
  - Build "recipe" scripts for diagnosing problems
  - Build "show" scripts for correlating data from multiple "show" commands
  - Build "action" scripts to perform common tasks

Network Troubleshooting

- **What are the steps in troubleshooting when a VPN is not functional ?**
- **Use Op Scripts**
  - Iteratively narrow down to the cause
  - Extend Tier-3 diagnostic expertise to Tier-1 operations

What Can you do with Op Scripts?

- **Op Script can:**
  - Run one or group of commands
  - Receive output in XML
  - Inspect output data; and
  - Determine next appropriate action
  - Repeat until source of problem is known
  - Problem can be reported to user via the CLI

Some Op Script Examples

- **Restarting an FPC with Slot number argument**
- **Display Domain Name System (DNS) information for a routing platform**
  - Do not need to enter a hostname or IP address for localhost
- **Customizing Output of the show interfaces**
- **Finding LSPs to Multiple Destinations**

Op Script Configuration

```
[system scripts op]
traceoptions {
    flag all;
}
file dead-peers.slax {
    description "Diagnose issues with dead peers";
    arguments {
        peer {
            description "Peer to diagnose";
        }
    }
}
file op-bchip.slax {
    description "B-Chip dump";
}
file op-host.xsl {
    description "simple reachability tests";
}
```

Op Script Execution

- **"op" command: op filename name1 val1 name2 val2**

```
user@host> op ?
Possible completions:
  <script>              Name of script to run
  dead-peers              Diagnose issues with dead peers
  op-bchip               B-Chip dump
  op-host                simple reachability tests
user@host> op dead-peers ?
Possible completions:
  <[Enter]>             Execute this command
  <name>                 Argument name
  detail                Display detailed output
  peer                  Peer to diagnose
  |                    Pipe through a command
user@host> op dead-peers peer 10.1.2.3
```

Session: show-dead-peers

```
user@host> op dead-peers peer 10.5.14.2
Peer: 10.5.14.2
Last error was: Cease
Last state was: OpenConfirm
...

user@host>
```

Agenda

- Introduction
- The High-Availability Imperative
- **Streamlining Network Operations**
  - Configuration Policy Enforcement
  - Automatic Configuration Generation
  - Custom Operational Commands
  - Automated Troubleshooting
- Summary

Event Policy Overview

- **User-customized policy for invoking an action upon detection of a specified event**
- **Accelerates detection and correction of error conditions**
  - Look at log messages
  - Correlate between log messages
  - Take action
    - Execute CLI command(s) and/or Op Script
    - Syslog Entry
    - Upload a File
    - Ignore the Event
- ***"If interface X went down and VPN Y went down, execute Op script & log customized message"***

Event Logic

- **"if"s are conditionals based on events**
  - event == syslog message
  - distinguished by "tag"
    - RPD_TASK_BEGIN, CHASSISD_FRU_EVENT
- **"then"s include:**
  - Ignore (end processing for the current event)
  - Execute JUNOS commands
    - Output is recorded and transferred to remote server
  - Upload specific files to remote server

Configuration (basic)

```
event-options {
  policy policy-name {
    events [ events ];
    then {
      execute-commands {
        commands {
          "command1";
          "command2";
        }
        output-filename filename;
        output-format (text | xml);
        destination dest-name;
      }
      ignore;
      upload filename fname destination dest-name;
    }
  }
}
```

Example: interface-up-down

```
event-options {
  policy save-if-data {
    events [ SMNP_TRAP_LINK_DOWN SNMP_TRAP_LINK_UP ];
    then {
      execute-commands {
        commands {
          "show interfaces";
          "show alarms";
        }
        output-filename if-status.txt;
        output-format text;
        destination my-server;
      }
    }
  }
}
```

Event Correlation

- **Detect connected events**
  - event ev1;
  - within 7200 event ev2;
- **Simple correlation**
  - Dampen events

```
event-options {
  policy dampen-policy {
    events [ ev1 ev2 ev3 ];
    within 3600 events [ ev4 ev5 ];
    then {
      ignore;
    }
  }
}
```

Example: ignore-maintenance

```
event-options {
  policy ignore-maintenance {
    events UI_COMMIT;
    within 7200 events MIDNIGHT;
    then {
      ignore;
    }
  }
  policy archive-config {
    events UI_COMMIT;
    then {
      execute-commands {
        commands {
          "configure";
          "status";
          "show | compare 1";
        }
      }
      ...
```

Destinations

- **Save output to local files**
- **Transfer files to remote servers**

```
[event-options]
destinations {
  name {
    transfer-delay seconds;
    archive-sites {
      url password password;
    }
  }
  my-server {
    archive-sites {
      ftp://nobody@my-server/log/data;
      nobody@my-server:log/data/;
    }
  }
}
```

Event Scripts

- **Event policies can call op scripts**
- **Leverage logic and extensibility of op scripts behind the power of events policies**

```
[event-options policy name then]
event-script filename.slax {
  arguments {
    name1 value1;
    name2 value2;
  }
  output-filename filename;
  destination dest-name;
}
```

JUNOScript Automation Best Practices

- Document Design Decisions in the Script
- Peer Review the Scripting Code
- Reuse Repository for Scripting Code

Agenda

- Introduction
- The High-Availability Imperative
- Streamlining Network Operations
  - Configuration Policy Enforcement
  - Automatic Configuration Generation
  - Custom Operational Commands
  - Automated Troubleshooting
- **Summary**

Benefits of Operational Automation

- Enforcement of compliance with standards and business policies
- Faster and Accurate device configurations
- Powerful Troubleshooting Tools
- Automated Diagnostics
- Increased Productivity and Network Availability

Q&A

Juniper your Net™