

Università degli Studi di Roma Tre

A Router Testing Framework For The Python Programming Language

Claudio Pisa
clauz at ninux.org

August 22, 2008

Abstract

A new framework for the Python Programming Language [1] is constructed from free, publicly available modules [2, 3, 4] and completed with components created ad-hoc. Then its use is shown by performing automated RFC 2328 (OSPF Version 2) [5] conformance tests on Juniper J2320 and Cisco 2811 routers.

Tools and methodologies used in this work could be enhanced, generalized and extended in order to obtain modular and router-independent automated tests.

Contents

1	Introduction	2
2	Building a Framework for Router Testing	2
2.1	Scapy	3
2.2	Pexpect	4
2.3	PySerial	4
2.4	Serialrouter, juniperj2320 and cisco2811	4
2.4.1	The SerialConnectedRouter class	5
2.4.2	The JuniperJ2320 class	5
2.4.3	The Cisco2811 class	7
2.5	Testsummary	8
2.6	Localconf	11
3	Tests	11
3.1	Basic Test	12
3.1.1	Basic Test on Juniper J2320	12
3.1.2	Basic Test on Cisco 2811	16
3.2	Adjacency Initial Forming Test	20
3.2.1	Adjacency Initial Forming Test on Juniper J2320	20
3.2.2	Adjacency Initial Forming Test on Cisco 2811	26

4 Conclusions and Further Work	28
4.1 Defects	28
4.2 Extensions	28
A Source Code of Some Framework Components	31
A.1 The serialrouter module	32
A.2 The juniperj2320 module	33
A.3 The cisco2811 module	35
A.4 The testsummary module	38
A.5 The localconf module	47
B Source Code of the Tests	47
B.1 The Basic Test	47
B.1.1 The Basic Test for Juniper J2320	47
B.1.2 The Basic Test for Cisco 2811	51
B.2 The Adjacency Initial Forming Test	55
B.2.1 The Adjacency Initial Forming Test for Juniper J2320	55
B.2.2 The Adjacency Initial Forming Test for Cisco 2811	59

1 Introduction

Open Shortest Path First (OSPF) [5] is a popular *interior gateway protocol* supported by most router vendors [6]. It is a *link-state* routing protocol: each router finds the best paths to other network destinations using a directed graph built from link-state advertisements (LSAs) received from other routers. For a more complete explanation please refer to the OSPF-related RFCs or to the available literature on the topic.

In this work, some tests are performed on commercial routers, specifically Juniper J2320 and Cisco 2811, to check the conformance with the OSPF Version 2 standard specified in RFC 2328 [5].

Section 3.1 shows a basic test in which in turn the routers are set up for OSPF operation, a connectivity test is performed using ICMP, and the correctness of emitted Hello packets is checked.

In section 3.2, the formation of an adjacency is started with the routers, using forged OSPF packets and obtaining Database Description packets that are checked for correctness.

The tests are automated through scripts written in the Python Programming Language [1], using a novel free and open source framework, described in section 2, constructed from libraries included with the Python standard distribution, some free modules publicly available on the Internet [2, 3, 4], and completed with the components whose source code is reported in appendix A.

Finally, in section 4, some possible enhancements and extensions of this work are illustrated.

2 Building a Framework for Router Testing

The automation of tests performed on router equipment may be helpful in several scenarios, e.g.:

- the same set of tests has to be performed on different routers, from the same vendor (e.g. defect detection tests) or from different vendors (e.g. standard conformance tests);
- regression tests, useful during the development of router operating systems.

The Python Programming language [1] may be an ideal candidate as the main tool for the achievement of this goal, since:

- allows quick development of programs and scripts;
- promotes the writing of code with a high level of readability, due to the mandatory use of indentation and the recommended use of inline documentation and of high-level constructs;
- supports object-oriented, modular programming;
- a large number of libraries are available, both in the standard Python distribution and over the Internet.

Furthermore, excluding tests in which speed is crucial (e.g. throughput measuring tests), and assuming that tests are ran from computers connected to the routers, the standard Python interpreter may be employed.

For a more exhaustive listing of Python's features please refer to Python's website [1].

In the remainder of this section, the components used to build the aforementioned testing framework are briefly described. In 2.1, 2.2 and 2.3 third-party modules are introduced, while in 2.4 and 2.5 the new free and open source modules created for this work are described.

2.1 Scapy

Scapy, by Philippe Biondi, [2] is a Python-based packet manipulation interactive program and a library, which provides an intuitive and rich API for multi-protocol packet forging, sending and capturing. OSPF support is not included natively, but an extension by Dirk Loss (*scapy_ospf*) is available on the Scapy wiki [7].

To illustrate the ease of its use, lets consider a simple script (§ listing 1) that forges an ICMP echo-request packet, sends it at a known network address and waits for an ICMP echo-reply.

Listing 1: A simple script using the scapy module

```

1  from scapy import *
2
3  # Forge an ICMP echo-request to the destination 192.168.1.1
4  p = IP(dst='192.168.1.1')/ICMP()
5  p[ICMP].type = 8 # echo-request
6
7  # Send the request and receive the reply
8  q = sr1(p)
9
10 q.show()

```

2.2 Pexpect

Pexpect, by Noah Spurrier and others [3], is a pure Python library in the spirit of Don Libes' Expect, a Unix automation and testing tool.

A program using pexpect should spawn a child process on which methods such as `expect()`, that waits for the appearance of a predefined pattern, and `sendline()`, that sends a string (e.g. a command) followed by a newline character, to the child process as if it was typed from a terminal, can be used. An example from the pexpect website follows, in listing 2.

Listing 2: A simple script using the pexpect module

```
# This connects to the openbsd ftp site and
# downloads the recursive directory listing.
4 import pexpect
child = pexpect.spawn ('ftp ftp.openbsd.org')
child.expect ('Name .*: ')
child.sendline ('anonymous')
child.expect ('Password: ')
child.sendline ('noah@example.com')
9 child.expect ('ftp> ')
child.sendline ('cd pub')
child.expect('ftp> ')
child.sendline ('get ls-1R.gz')
child.expect('ftp> ')
14 child.sendline ('bye')
```

An extension called *fdpexpect* allows to attach the same methods to any file descriptor, associated to open files or character devices.

2.3 PySerial

PySerial, by Chris Liechti [4], provides system-independent encapsulated access to serial ports. Supports file-like API, various serial port connection parameters and binary transmission (i.e. no character translation).

For example, to connect to the serial device `/dev/ttyS0` with a speed of 9600 bps, no parity bit, 8 data bits per character, and one stop bit, the code reported in listing 3 could be used.

Listing 3: A simple script using the pySerial module

```
import serial

s = serial.serialposix.Serial('/dev/ttyS0', baudrate = 9600, \
    bytesize = 8, parity='N', stopbits=1)
5 s.write("ATZ\r\n")
readok = s.read(2)
10 s.close()
```

2.4 Serialrouter, juniperj2320 and cisco2811

The idea behind the `serialrouter` module is that combining the `serial.serialposix.Serial` class (§ section 2.3) with the `fdpexpect.spawn` class (§ section 2.2), an API for the control of a router connected through a serial port may be obtained.

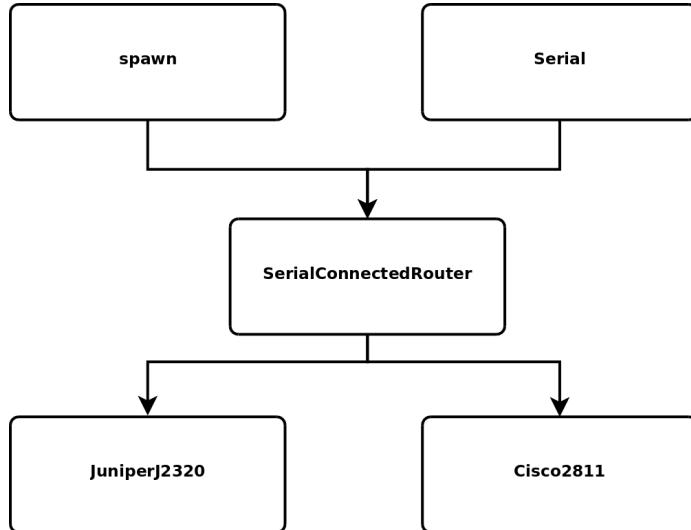


Figure 1: Class derivation diagram.

For this purpose, the `serialrouter.SerialConnectedRouter` class is defined, from which the router-dependent `juniperj2320.JuniperJ2320` and `cisco2811.Cisco2811` classes are derived (§ figure 1).

An overview of these classes follows. For a more in-depth description, please use the Pydoc documentation¹ or refer to the source code, reported in sections A.1, A.2 and A.3.

2.4.1 The `SerialConnectedRouter` class

The `serialrouter.SerialConnectedRouter` class provides an API to control a router connected through a serial port.

As previously stated, it is derived, in this order, from the `serial.serialposix.Serial` and the `fdpexpect.spawn` classes. Python's rule for resolving class attribute references is "depth-first, left-to-right", thus some methods defined in `fdpexpect.spawn`, like `read()` or `write()`, are overridden by the homonymous methods defined in `serial.serialposix.Serial`. Moreover the method `serialrouter.SerialConnectedRouter.sendline()` overrides `fdpexpect.spawn.sendline()`.

The class is not operating system-dependent, as it uses components available only on POSIX [8] compliant systems.

2.4.2 The `JuniperJ2320` class

The `juniperj2320.JuniperJ2320` class is a direct descendent of the `serialrouter.SerialConnectedRouter` class. It provides quick access to the specific features of a Juniper J2320 router connected through a serial port

¹ The Pydoc documentation is usually accessible on systems where Python is installed by typing `pydoc <module name>` (e.g. `pydoc testsummary`) on the command line.

and using the JUNOS operating system². A summary of the methods of the `juniperj2320.JuniperJ2320` class follows.

- **instantiation:** when creating a new instance of the `JuniperJ2320` class, the serial device must be specified as the constructor argument; e.g.:

```
from juniperj2320 import *
router = JuniperJ2320('/dev/ttyUSB0')
```

- **setUsername() and setPassword():** specify the username and password needed to log into the router.
- **gotologinscreen():** climb the JUNOS configuration hierarchy until the login prompt appears.
- **login():** actually log into the router; e.g.:

```
1   router.setUsername('root')
2   router.setPassword('secret')
3   router.gotologinscreen()
4   router.login()
```

- **sendcommand():** send a command to the router using the serial port.
- **timedexpect():** wait for the appearance of a pattern on the serial port and raise an exception if timeout occurs; e.g.:

```
1   router.sendcommand('delete interfaces ge-0/0/0 unit 0 family inet')
2   # wait for the configuration prompt
3   router.timedexpect('#')
```

- **clipromptexpect():** wait for the appearance of the command line interface (CLI) prompt (i.e. ">").
- **gotocli():** go to CLI mode, i.e. climb or descend the JUNOS configuration hierarchy until the CLI prompt appears.
- **confpromptexpect():** wait for the appearance of the configuration prompt (i.e. "#").
- **gotoconf():** go to configuration mode, i.e. climb or descend the JUNOS configuration hierarchy until the configuration prompt appears.
- **commit():** send a “commit” command to the router, which must be in configuration mode, and wait for the commit to complete or, if timeout occurs, raise an exception; e.g.:

```
1   router.gotoconf()
2   router.sendcommand('delete interfaces ge-0/0/0 unit 0 family inet')
3   router.confpromptexpect()
4   router.commit()
```

- **readuntil():** reads from the serial device until the specified pattern is met; e.g.:

²The `juniperj2320` module has been tested only on a Juniper J2320 router with JUNOS Software Release [8.4R1.13] (Export edition), but should work with other router models and JUNOS releases as well.

```

1  router.gotocli()
2  router.clipromptexpect()
3  router.sendcommand("show version")
4
5  # jump, in the input stream, after the echo
6  # of the command that we just sent
7  router.readuntil('\n')
8
9  routerhostname = router.readuntil('\n')
10 routermodel = router.readuntil('\n')
11 routeros = router.readuntil('\n')

```

2.4.3 The Cisco2811 class

The `cisco2811.Cisco2811` class is also a direct descendent of the `serialrouter.SerialConnectedRouter` class. Similar to `juniperJ2320.JuniperJ2320`, provides quick access to the features of a Cisco 2811 router connected through a serial port and running the Cisco IOS operating system³. The main methods of the class are listed below.

- **instantiation:** when creating a new instance of the `Cisco2811` class, the serial device must be specified as the constructor argument; e.g.:

```

from cisco2811 import *
router = Cisco2811('/dev/ttyUSB0')

```

- **setUsername() and setPassword():** specify the username and password needed to log into the router.
- **setHostname():** specify the host name. This is needed in order to match the router prompts more closely in the `clipromptexpect()`, `enabledpromptexpect()`, `configpromptexpect()` and `config_promptexpect()` methods.
- **gotologinscreen():** climb the Cisco IOS configuration hierarchy until the login prompt appears.
- **login():** actually log into the router; e.g.:

```

3  router.setUsername('admin')
   router.setPassword('secret')
   router.setHostname('cisco2')
   router.gotologinscreen()
   router.login()

```

- **sendcommand():** send a command to the router using the serial port.
- **timedexpect():** wait for the appearance of a pattern on the serial port and raise an exception if timeout occurs; e.g.:

```

router.sendcommand('terminal length 0')
# wait for the '>' prompt
router.timedexpect('cisco2>')

```

³The `cisco2811` module has been tested only on a Cisco 2811 router with Cisco IOS Software, 2800 Software (C2800NM-ADVIPSERVICESK9-M), Version 12.4(9)T6, RELEASE SOFTWARE (fc2) on board, but should be also compatible with other router models and Cisco IOS versions.

- **clipromptexpect()**: wait for the appearance of the ">" prompt.
- **gotocli()**: go to the initial command line interface (CLI) prompt, i.e. climb or descend the Cisco IOS configuration hierarchy until the ">" prompt appears.
- **enabledpromptexpect()**: wait for the appearance of the privileged EXEC mode prompt (i.e. "#").
- **gotoenabled()**: go to privileged EXEC mode, i.e. climb or descend the Cisco IOS configuration hierarchy, entering the password where appropriate, until the "#" prompt appears.
- **configpromptexpect()**: waits for the appearance of the global configuration mode prompt (i.e. "(config)#").
- **gotoconfig()**: go to global configuration mode, i.e. climb or descend the Cisco IOS configuration hierarchy until the "(config)" prompt appears.
- **config_promptexpect()**: waits for the appearance of a configuration mode prompt (e.g. "(config-router)#").
- **write()**: go to privileged EXEC mode and issue the write command on the router, in order to save the current configuration; e.g.:

```

2   # go to configuration mode logging in if needed
3   router.gotoconfig()
4   router.sendcommand("interface FastEthernet 0/0")
5   # wait for the '(config-if)#' prompt
6   router.config_promptexpect("if")
7   router.sendcommand("ip address 191.168.0.31 255.255.255.0")
8   router.config_promptexpect("if")
9   router.sendcommand("no shutdown")
10  router.config_promptexpect("if")
11  router.sendcommand("end")
12  router.enabledpromptexpect()
13  router.write()

```

- **readuntil()**: reads from the serial device until the specified pattern is met; e.g.:

```

3   router.gotocli()
4   router.clipromptexpect()
5   router.sendcommand("show version")

# jump in the input stream, after the echo
# of the command that we just sent
6   router.readuntil('\n')
7
8   routervinfo = router.readuntil(router.cliprompt)
9   print routervinfo

```

2.5 Testsummary

The testsummary module provides an API to manage test runs and store related results.

Similar modules, like unittest, included in the standard Python distribution, or UTscapy, which can be found on the Scapy website [2], are focused on software tests, and thus not suited for the goals of this work, so a new module is built ad-hoc from scratch, and its source code is reported in appendix A.4.

The usage of this module is now introduced. Its main class is `testsummary.Test`, which represents a *test run*. Each test is composed by several *subtests*, which in turn may include various *results*.

- **instantiation:** when creating a new test object, its title must be specified to the constructor; e.g.:

```
from testsummary import *
test = Test("Foo Bar")
```

- **addSubtest():** creates a new *subtest* in the test by specifying a label, which may be an integer or a string. Moreover, a parameter may specify if the subtest is a *task*⁴ (e.g. performs initial setup before the “real” subtests); e.g.:

```
# In this example three subtests are created.
# The first subtest is a task
3 test.addSubtest('initial configuration', task = True)

# The second subtest is the first "real" subtest
test.addSubtest('first subtest')

8 # The third subtest is the second "real" subtest
# and is specified with an integer label
test.addSubtest(2)
```

- **addSubtestTitle():** gives a title to a previously created subtest; e.g.:

```
test.addSubtest(3)
test.addSubtestTitle(3, "The third subtest")
```

- **addSubtestDependency():** specifies that a subtest depends on the success of another subtest; e.g.:

```
# Subtest 3 depends on the success of subtest
# 'initial configuration' and of subtest 2
3 test.addSubtestDependency(3, 'initial configuration')
test.addSubtestDependency(3, 2)
```

- **addResult():** adds a result to a subtest. The first parameter is the subtest label, the second the description of the result and the third the value; e.g.:

```
1 test.addResult('first subtest', "Router OS Version", "JUNOS 8.4")
```

- **begin():** should be executed as the first command at the beginning of a subtest. Checks that all subtest dependencies are met, and if not, raises a `TestDependencyException` exception.

- **end():** is used to specify the final result of a subtest. Its predefined values are: `TEST_OK`, if the test succeeded, `TEST_FAILED`, if the test failed, or `TEST_SKIPPED`, if the test was skipped, for example due to a dependency issue; e.g.:

⁴ At the moment, the only differences between a normal subtest and a subtest marked as being a task are that, in case of failure (`TEST_FAILED`), when the test summary is printed, for a normal subtest the string “FAILED” is printed, while for a task, the string “ERROR” is printed. In case of success (`TEST_OK`), the “PASSED” or “DONE” strings are printed for normal subtests or tasks, respectively.

```

# ...
import traceback
# ...
# ...

4   try:
    test.begin(3)

9     # perform the test, eventually using assert statements
# ...
# ...
    test.addResult(3, "Router model", routermode)

14  except TestDependencyException:
    # The dependencies were not met
    test.end(3, TEST_SKIPPED)
except Exception, err:
    # An error occurred
    print type(err), err
    traceback.print_tb(sys.exc_info() [2])
    test.end(3, TEST_FAILED)
except:
    # Unexpected error
    raise
else:
    # The test succeeded
    test.end(3, TEST_OK)

```

- **printTitleString()** and **announce()**: both methods are used to print messages on the screen, with decreasing level of importance.
- **Test's string representation:** by using the `str()` or function or the `print` command on a Test object, a textual summary of the test may be obtained; e.g. the `print test` command may print the following:

```

*****
***** Foo Bar *****
*****
=====
initial configuration      DONE
=====
first subtest              FAILED
=====
Router OS Version          JUNOS 8.4
=====
2                          PASSED
=====
The third subtest          PASSED
=====
Router model                Juniper J2320
=====

*****

```

- **getTeX()**: this method returns a table with the summary of the test using `\TeX` syntax; e.g. (rendered):

Foo Bar	
initial configuration	DONE
first subtest	FAILED
Router OS Version	JUNOS 8.4
2	PASSED
The third subtest	PASSED
Router model	Juniper J2320

- **save()**: saves the Test object on a file (using the `pickle` module). If no filename is specified, it is obtained from the test's title and the current date and time.

- **testload()**: not a method of the Test class, loads a saved test summary from a file; e.g.:

```
# save the file in the directory /tmp using a filename automatically
# created from current time and date
3  savedfile = test.save(dir = "/tmp")

# destroy the test object
del test

8 # and load it again
test = testload(savedfile)
```

2.6 Localconf

The localconf module is used to issue commands to the local system. Its (short) source code is reported in appendix A.5. A single function is defined in localconf:

- **localcommand()**: executes a local command; e.g.:

```
1 localconf.localcommand("ip addr flush dev eth0")
```

3 Tests

In this section the framework described in section 2 is used to perform RFC 2328 (OSPF Version 2) [5] compliance tests on Juniper J2320 and Cisco 2811 routers.

The tests, called *Basic Test* and *Adjacency Initial Forming Test*, are ran on each router separately, for a total of four tests. Each router in turn is connected to a computer running the GNU/Linux operating system using both a serial cable⁵ and an Ethernet cable (§ figure 2).

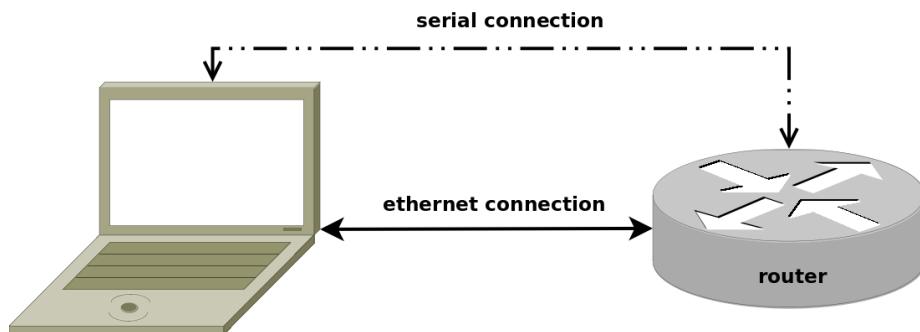


Figure 2: Connections between computer and router equipment for test execution.

Furthermore, a sniffer may be started in parallel on the local machine, in order to monitor the tests while they are ran.

⁵On modern laptops, where no serial ports are available, an USB to serial adapter may be used. In this case the Linux `usbserial` module could be useful.

3.1 Basic Test

This section shows a test called *Basic Test* in which:

- a router is configured for OSPF version 2 operation;
- router information is retrieved;
- a connectivity test is performed using ICMP echo-request and echo-reply packets;
- an emitted OSPF Hello packet is captured and checked for correctness;
- a final setup is performed.

The test is performed at first on the Juniper J2320 router and then repeated, with some necessary changes in the initial and final setup, on the Cisco 2811 router.

The code of the tests is fragmented for a clearer exposition. For complete, unfragmented listings, please refer to appendix [B.1](#).

3.1.1 Basic Test on Juniper J2320

In this section the code of the Basic Test is explained, and then the results of its running with a connected Juniper J2320 router are shown. The integral code listing will be referred to as “listing 9”, and can be found in appendix [B.1.1](#).

As usual for Python programs, the first lines are dedicated to import statements:

```
# Perform an ICMP connectivity test and verify the emission
# of correct OSPF Hello packets from a Juniper J2320 router

4  from scapy_ospf import *
from localconf import *
from juniperj2320 import *
from testsummary import *
import time
```

Except for `time`, which is included in the standard Python distribution, the included modules are described above in section [2](#).

Then some self-explaining local constants are defined:

```
SERIALDEVICE = '/dev/ttyUSB0'
ROUTER_IP = '191.168.0.31'
ROUTER_MASK = '24'
ROUTER_USERNAME = 'root'
ROUTER_PASSWORD = 'secret'
ROUTER_INTERFACE = "ge-0/0/0"
16 LOCAL_INTERFACE = 'eth0'
LOCAL_IP = '191.168.0.32'
LOCAL_MASK = '24'
OSPF_AREA = '0.0.0.0'

21 TEST_OUTPUT_DIR = "./test-runs"
```

The creation of a `Test` object and the definition of subtests and their dependencies follows (§ section [2.5](#)):

```
test = Test("Juniper J2320 Basic Test")

# The subtests
26 test.addSubtest('localconf', task = True)
test.addSubtestTitle('localconf', "Local setup")

test.addSubtest('routerconf', task = True)
```

```

31 test.addSubtestTitle('routerconf', "Router setup")
# if local set-up was not successful do not configure the router
32 test.addSubtestDependency('routerconf', 'localconf')

33 test.addSubtest('routerinfo', task = True)
34 test.addSubtestTitle('routerinfo', "Retrieve router model information")
35 test.addSubtestDependency('routerinfo', 'routerconf')

36 test.addSubtest('icmp')
37 test.addSubtestTitle('icmp', "ICMP connectivity test")
38 test.addSubtestDependency('icmp', 'localconf')
39 test.addSubtestDependency('icmp', 'routerconf')

40 test.addSubtest('hello')
41 test.addSubtestTitle('hello', "Emission of correct OSPF Hello packets")
42 test.addSubtestDependency('hello', 'localconf')
43 test.addSubtestDependency('hello', 'routerconf')

44 test.addSubtest('ospfdisable', task = True)
45 test.addSubtestTitle('ospfdisable', "Disable OSPF on the router")
46 test.addSubtestDependency('ospfdisable', 'routerconf')

```

In the first subtest, defined as a task on line 26, an IP address for the local GNU/Linux system is configured, using the localconf module (§ section 2.6) and the *iproute* [9] command. Then the configuration parameters are saved as results of the subtest using the addResult() method:

```

# Local machine configuration
try:
    test.begin('localconf')
    localcommand("ip addr flush dev %s" % LOCAL_INTERFACE)
    localcommand("ip addr add %s/%s dev %s" % (LOCAL_IP, LOCAL_MASK,
                                                LOCAL_INTERFACE))
    test.addResult('localconf', "Local IP Address", LOCAL_IP)
    test.addResult('localconf', "Local Netmask", "/" + LOCAL_MASK)
    localcommand("ip link set %s up" % LOCAL_INTERFACE)
    test.addResult('localconf', "Local Interface", LOCAL_INTERFACE)
except:
    test.end('localconf', TEST_FAILED)
else:
    test.end('localconf', TEST_OK)

```

The router configuration subtest (defined as a task too) comes next. A JuniperJ2320 object is instantiated (§ section 2.4.2), a logfile is opened for debugging purposes, and IP and OSPF configuration is performed using JUNOS-specific commands:

```

# Now configure the router
try:
    test.begin('routerconf')
    router = JuniperJ2320(SERIALDEVICE)

    # turn on logging
    71 logfile = open("%s/juniperj2320-%s.log" % (TEST_OUTPUT_DIR, time.time()),
                     "w")
    router.logfile = logfile

    76 router.setUsername(ROUTER_USERNAME)
    router.setPassword(ROUTER_PASSWORD)

    test.addResult('routerconf', "Serial Device", SERIALDEVICE)
    test.addResult('routerconf', "Router Username", ROUTER_USERNAME)
    test.addResult('routerconf', "Router Password", "***")

    81 router.gotoconf()
    router.sendcommand("delete interfaces %s unit 0 family inet" %
                      ROUTER_INTERFACE)
    router.confpromptexpect()
    router.sendcommand("set interfaces %s unit 0 family inet address %s/%s" \

```

```

    % (ROUTER_INTERFACE, ROUTER_IP, ROUTER_MASK)

# OSPF configuration
91 router.confpromptexpect()
router.sendcommand("set routing-options router-id %s" % ROUTER_IP)
router.confpromptexpect(timeout=10)
router.sendcommand("set protocols ospf area %s interface %s enable" \
    % (OSPF_AREA, ROUTER_INTERFACE))
router.confpromptexpect(timeout=10)
router.sendcommand("set protocols ospf enable")

# commit
96 router.commit()

101 test.addResult('routerconf', "Router Interface", ROUTER_INTERFACE)
test.addResult('routerconf', "Router IP Address", ROUTER_IP)
test.addResult('routerconf', "Router Netmask", "/" + ROUTER_MASK)
test.addResult('routerconf', "OSPF Area", OSPF_AREA)

106 except TestDependencyException:
    # The dependencies were not met
    test.end('routerconf', TEST_SKIPPED)
except Exception, err:
    # An error occurred
    111 print type(err), err
    test.end('routerconf', TEST_FAILED)
except:
    # Unexpected error
    raise
116 else:
    # The test succeeded
    test.end('routerconf', TEST_OK)

```

A simple subtest/task, labeled 'routerinfo', retrieves router information from the router (§ 2.4.2) by using the show version command and then stores it as a result of the subtest:

```

try:
123 test.begin('routerinfo')

    router.gotocli()
    router.clipromptexpect()
    router.sendcommand("show version")
    router.readuntil('\n')

    routerhostname = router.readuntil('\n')
    print routerhostname

    routermode = router.readuntil('\n')
    print routermode

    routeros = router.readuntil('\n')
    print routeros

138 test.addResult('routerinfo', "Router Hostname", routerhostname)
test.addResult('routerinfo', "Router Model", routermode)
test.addResult('routerinfo', "Router OS", routeros)

143 except TestDependencyException:
    # The dependencies were not met
    test.end('routerinfo', TEST_SKIPPED)
except Exception, err:
    # An error occurred
    148 print type(err), err
    test.end('routerinfo', TEST_FAILED)
except:
    # Unexpected error
    raise
153 else:
    # The test succeeded
    test.end('routerinfo', TEST.OK)

```

In the 'icmp' subtest, the scapy module (§ section 2.1) is used to forge an ICMP echo-request packet, and wait for an ICMP echo-reply packet from the router:

```
# now check connectivity using icmp
try:
    test.begin('icmp')
    test.announce("Checking connectivity using ICMP")
162
    conf.iface = LOCAL_INTERFACE

    # an icmp echo-request packet
    icmp_echo_request = IP(dst=ROUTER_IP)/ICMP() / "XXXXXXXXXXXXXXXXXXXX"
167
    print "Sending an ICMP echo-request packet"
    assert(icmp_echo_request != None)

172
    icmp_echo_request.show()

    # send the packet and get the reply
    icmp_echo_reply = sr1(icmp_echo_request, timeout = 10)

    assert(icmp_echo_reply != None)
    print "ICMP echo-reply received"
    icmp_echo_reply.show()

    assert(icmp_echo_reply.type == 0)
182
except TestDependencyException:
    # The dependencies were not met
    test.end('icmp', TEST_SKIPPED)
except Exception, err:
    print type(err), err
    test.end('icmp', TEST_FAILED)
except:
    # Unexpected error
    raise
else:
    # The test succeeded
    test.end('icmp', TEST_OK)
```

Scapy's `sr1()` function is used to send a packet and wait for the related response (in this case to send an ICMP echo-request packet and wait for an ICMP echo-reply packet). Moreover some assert statements are executed, in order to raise an exception, and thus fail the test, if the asserted expressions are evaluated as false. If such an exception is raised, the `except` statement on line 186 catches it and associates the `TEST_FAILED` final result to the subtest.

Subsequently, using Scapy's `sniff()` function, an OSPF Hello packet from the router is sniffed and its correctness is asserted:

```
# Now sniff an ospf hello packet
try:
    test.begin('hello')
    test.announce("Trying to sniff an OSPF Hello Packet...")
197
    sniffedpackets = sniff(count=1, lfilter = lambda x: x.haslayer(OSPF_Hello),
                           timeout=60)
    assert(len(sniffedpackets) > 0)
    sniffedpackets.show()
    p = sniffedpackets[0]
    pospf = p.getlayer(OSPF_Hdr)
    pospf.display()
207
    test.addResult('hello', 'OSPF Type', pospf.type)
    test.addResult('hello', 'OSPF Version', pospf.version)
    test.addResult('hello', 'OSPF Source address', pospf.src)
    test.addResult('hello', 'OSPF Area', pospf.area)
    test.addResult('hello', 'OSPF Auth Type', pospf.authtype)
    test.addResult('hello', 'OSPF Hello Interval', pospf.hellointerval)
    test.addResult('hello', 'OSPF Hello Dead Interval', pospf.deadinterval)
212
```

```

217     test.addResult('hello', 'OSPF Hello Options', pospf.options)
218     test.addResult('hello', 'OSPF Hello NetMask', pospf.mask)
219     test.addResult('hello', 'OSPF Hello Designated Router', pospf.router)
220     test.addResult('hello', 'OSPF Hello Backup Router', pospf.backup)
221     test.addResult('hello', 'OSPF Hello Neighbors', pospf.neighbor)
222     assert(pospf.type == 1)
223     assert(pospf.version == 2)
224     assert(pospf.src == ROUTER_IP)
225     assert(pospf.area == OSPF_AREA)

226 except TestDependencyException:
227     # The dependencies were not met
228     test.end('hello', TEST_SKIPPED)
229 except Exception, err:
230     # An error occurred
231     print type(err), err
232     test.end('hello', TEST_FAILED)
233 except:
234     # Unexpected error
235     raise
236 else:
237     # The test succeeded
238     test.end('hello', TEST_OK)

```

Then OSPF is disabled on the router:

```

# now disable ospf on the router
242 try:
243     test.begin('ospfdisable')
244     router.gotoconf()
245     router.confpromptexpect()
246     router.sendcommand("set protocols ospf disable")
247     router.commit()
248     router.gotologinscreen()

249 except TestDependencyException:
250     # The dependencies were not met
251     test.end('ospfdisable', TEST_SKIPPED)
252 except Exception, err:
253     # An error occurred
254     print type(err), err
255     test.end('ospfdisable', TEST_FAILED)
256 except:
257     # Unexpected error
258     raise
259 else:
260     # The test succeeded
261     test.end('ospfdisable', TEST_OK)

```

And, finally, after closing the logfile used for debugging purposes, the results of the test are displayed and saved.

```

# turn off logging
logfile.close()
266 print test
test.save(dir = TEST_OUTPUT.DIR)

```

Running the above-explained test on a Juniper J2320 router (connected as described in figure 2) the test summary reported in table 1 was obtained.

3.1.2 Basic Test on Cisco 2811

In order to run the Basic Test for the Juniper J2320 router described in the previous section with a Cisco 2811 router, some minor changes have to be implemented. The complete source code can be found in appendix B.1.2.

The initial import statements now include the cisco2811 module:

Juniper J2320 Basic Test	
Local setup	DONE
Local IP Address	191.168.0.32
Local Netmask	/24
Local Interface	eth0
Router setup	DONE
Serial Device	/dev/ttyUSB0
Router Username	root
Router Password	***
Router Interface	ge-0/0/0
Router IP Address	191.168.0.31
Router Netmask	/24
OSPF Area	0.0.0.0
Retrieve router model information	DONE
Router Hostname	Hostname: j2320
Router Model	Model: j2320
Router OS	JUNOS Software Release [8.4R1.13] (Export edition)
ICMP connectivity test	PASSED
Emission of correct OSPF Hello packets	PASSED
OSPF Type	1
OSPF Version	2
OSPF Source address	191.168.0.31
OSPF Area	0.0.0.0
OSPF Auth Type	0
OSPF Hello Interval	10
OSPF Hello Dead Interval	40
OSPF Hello Options	2
OSPF Hello NetMask	255.255.255.0
OSPF Hello Designated Router	0.0.0.0
OSPF Hello Backup Router	0.0.0.0
OSPF Hello Neighbors	0.0.0.0
Disable OSPF on the router	DONE

Table 1: The summary of the Basic Test performed on a Juniper J2320 router.

```

1 # Perform an ICMP connectivity test and verify the emission
# of correct OSPF Hello packets from a Juniper J2320 router

from scapy_ospf import *
from localconf import *
from cisco2811 import *
from testsummary import *
import time

```

Netmasks have to be specified in dotted decimal notation, and also the name of the interface on the router is different:

```

11 SERIALDEVICE = '/dev/ttyUSB0'
ROUTER_IP = '191.168.0.31'
ROUTER_MASK = '255.255.255.0'
ROUTER_USERNAME = 'admin'
ROUTER_PASSWORD = 'secret'
ROUTER_HOSTNAME = 'cisco2'
16 ROUTER_INTERFACE = "FastEthernet 0/0"
LOCAL_INTERFACE = 'eth0'
LOCAL_IP = '191.168.0.32'
LOCAL_MASK = '24'
OSPF_AREA = '0.0.0.0'
21 TEST_OUTPUT_DIR = "./test-runs"

```

In the instantiation of the Test object, an appropriate title is passed as the argument:

```
test = Test("Cisco 2811 Basic Test")
```

And then the same subtest definitions and dependencies used in listing 9, are used, and thus are here omitted. Modifications are neither needed for the local configuration ('localconf') subtest.

On the contrary, some changes are needed for the router configuration ('routerconf') and router information retrieving ('routerinfo') subtests:

```

68 # Now configure the router
try:
    test.begin('routerconf')
    router = Cisco2811(SERIALDEVICE)

73     # turn on logging
    logfile = open("%s/cisco2811-%s.log" % (TEST_OUTPUT_DIR, time.time()), "w")
    router.logfile = logfile

    router.setUsername(ROUTER_USERNAME)
    router.setPassword(ROUTER_PASSWORD)
    router.setHostname(ROUTER_HOSTNAME)

    test.addResult('routerconf', "Serial Device", SERIALDEVICE)

83    router.gotoconfig()
    test.addResult('routerconf', "Router Username", ROUTER_USERNAME)
    test.addResult('routerconf', "Router Password", "***")

    router.gotoconfig()
88    router.sendcommand("interface %s" % ROUTER_INTERFACE)
    router.config_promptexpect("if")
    router.sendcommand("ip address %s %s" % (ROUTER_IP, ROUTER_MASK))
    router.config_promptexpect("if")
    router.sendcommand("no shutdown")
    router.config_promptexpect("if")
    router.sendcommand("end")
    router.enabledpromptexpect()

93    # OSPF configuration
    router.gotoconfig()

```

```

    router.sendcommand("router ospf 100")
    router.config.promptexpect("router")
    router.sendcommand("network %s 255.255.255.255 area %s" %(ROUTER_IP,
        OSPF_AREA))
    router.config.promptexpect("router")
    router.sendcommand("end")

    # write configuration
    router.write()

108    test.addResult('routerconf', "Router Interface", ROUTER_INTERFACE)
    test.addResult('routerconf', "Router IP Address", ROUTER_IP)
    test.addResult('routerconf', "Router Netmask", "/" + ROUTER_MASK)
    test.addResult('routerconf', "OSPF Area", OSPF_AREA)

113    except TestDependencyException:
        # The dependencies were not met
        test.end('routerconf', TEST_SKIPPED)
    except Exception, err:
        # An error occurred
        print type(err), err
        test.end('routerconf', TEST_FAILED)
    except:
        # Unexpected error
        raise
123    else:
        # The test succeeded
        test.end('routerconf', TEST_OK)

128    # Retrieve router information
try:
    test.begin('routerinfo')

    router.gotocli()
    router.clipromptexpect()
    router.sendcommand("show version")
    router.readuntil('\n')

    routerinfo = router.readuntil(router.cliprompt)
    print routerinfo

    routerinfo = "\n" + routerinfo

    test.addResult('routerinfo', "Router Information", routerinfo)
143
except TestDependencyException:
    # The dependencies were not met
    test.end('routerinfo', TEST_SKIPPED)
except Exception, err:
    # An error occurred
    print type(err), err
    test.end('routerinfo', TEST_FAILED)
except:
    # Unexpected error
    raise
153    else:
        # The test succeeded
        test.end('routerinfo', TEST_OK)

```

For the central tests, i.e. the ICMP connectivty ('icmp') and the OSPF Hello correctness ('hello') tests, no changes are needed at all.

But disabling OSPF on the router requires a slight modification:

```

try:
    test.begin('ospfdisable')
    router.gotoconfig()
    router.sendcommand("no router ospf 100")
    router.configpromptexpect()
    # Write configuration
    router.write()
    router.gotologinscreen()
246
251

```

```

except TestDependencyException:
    # The dependencies were not met
    test.end('ospfdisable', TEST_SKIPPED)
except Exception, err:
    # An error occurred
    print type(err), err
    test.end('ospfdisable', TEST_FAILED)
except:
    # Unexpected error
    raise
else:
    # The test succeeded
    test.end('ospfdisable', TEST_OK)

```

Running this test on a Cisco 2811 router yielded the results summarized in table 2.

3.2 Adjacency Initial Forming Test

In the OSPF Version 2 protocol, *adjacencies* between routers are formed in order to permit the exchange of routing information. During the formation of an adjacency various types of packets are exchanged: Hello, Database Description, Link State Request and, finally, Link State Advertisement packets. In this test only Hello and Database Description packets are exchanged, and thus the adjacency forming process is only initiated. An example of the complete process may be found in section 10.10 of [5].

In this section the *Adjacency Initial Forming Test* is described, where:

- a router is configured for OSPF version 2 operation;
- an emitted OSPF Hello packet is captured and checked for correctness;
- an OSPF Hello packet with high *priority* value is forged and sent to the router;
- an emitted OSPF Database Description packet is captured and checked for correctness;
- a final setup is performed.

The test is performed on the Juniper J2320 router and then repeated on the Cisco 2811 router with some necessary differences, but leaving unaltered the central part of the script.

The code of the tests is fragmented for a clearer exposition. For complete, unfragmented listings, please refer to appendix B.2.

3.2.1 Adjacency Initial Forming Test on Juniper J2320

This section explains the source code of the Adjacency Initial Forming Test and shows the results of its execution with a connected Juniper J2320 router. The integral code listing will be referred to as “listing 11”, and can be found in appendix B.2.1.

After the initial import statements, some constants are defined:

```

1  from scapy_ospf import *
from localconf import *
from juniperj2320 import *
from testsummary import *
import sys
import traceback

# Verify that the router behaves as in section 10.10 of RFC 2328 (OSPFv2),
# where an adjacency forming example is shown

```

Cisco 2811 Basic Test	
	Local setup
Local IP Address	191.168.0.32
Local Netmask	/24
Local Interface	eth0
	Router setup
Serial Device	/dev/ttyUSB0
Router Username	admin
Router Password	***
Router Interface	FastEthernet 0/0
Router IP Address	191.168.0.31
Router Netmask	/255.255.255.0
OSPF Area	0.0.0.0
	Retrieve router model information
Router Information	Cisco IOS Software, 2800 Software (C2800NM-ADVIPSERVICESK9-M), Version 12.4(9)T6, RELEASE SOFTWARE (fc2) Technical Support: http://www.cisco.com/techsupport Copyright (c) 1986-2007 by Cisco Systems, Inc. Compiled Thu 18-Oct-07 18:01 by prod_rel_team ROM: System Bootstrap, Version 12.4(13r)T, RELEASE SOFTWARE (fc1) cisco2 uptime is 1 hour, 49 minutes System returned to ROM by power-on System image file is "flash:c2800nm-advipservicesk9-mz.124-9.T6.bin" [...] Cisco 2811 (revision 53.51) with 249856K/12288K bytes of memory. Processor board ID FCZ1203715E 2 FastEthernet interfaces 1 Virtual Private Network (VPN) Module DRAM configuration is 64 bits wide with parity enabled. 239K bytes of non-volatile configuration memory. 62720K bytes of ATA CompactFlash (Read/Write) Configuration register is 0x3922
	ICMP connectivity test
Emission of correct OSPF Hello packets	PASSED
OSPF Type	1
OSPF Version	2
OSPF Source address	191.168.0.31
OSPF Area	0.0.0.0
OSPF Auth Type	0
OSPF Hello Interval	10
OSPF Hello Dead Interval	40
OSPF Hello Options	18
OSPF Hello NetMask	255.255.255.0
OSPF Hello Designated Router	191.168.0.31
OSPF Hello Backup Router	0.0.0.0
OSPF Hello Neighbors	None
	Disable OSPF on the router
	DONE

Table 2: The summary of the Basic Test performed on a Cisco 2811 router.

```

11 AllSPFRouters = '224.0.0.5'
12
13 SERIALDEVICE = '/dev/ttyUSB0'
14 ROUTER_IP = '191.168.0.31'
15 ROUTER_MASK = '24'
16 ROUTER_USERNAME = 'root'
17 ROUTER_PASSWORD = 'secret'
18 ROUTER_INTERFACE = 'ge-0/0/0'
19 LOCAL_INTERFACE = 'eth0'
20 LOCAL_IP = '191.168.0.32'
21 LOCAL_MASK = '24'
22 LOCAL_FULL_MASK = '255.255.255.0'
23 OSPF_AREA = '0.0.0.0'
24 ROUTER_PRIORITY = 100
25 LOCAL_PRIORITY = 200
26 TEST_RUN_DIR = "./test-runs"

```

The AllSPFRouters multicast address' value is defined in appendix A.1 of [5]. Then a Test object is instantiated and the subtests and their dependencies are defined:

```

test = Test("Juniper J2320 RFC2328 Section 10.10 Example Conformance")
31 test.addSubtest('localconf', task = True)
32 test.addSubtestTitle('localconf', "Local setup")
33
34 test.addSubtest('routerconf', task = True)
35 test.addSubtestTitle('routerconf', "Router setup")
36 # if local set-up was not successful do not configure the router
37 test.addSubtestDependency('routerconf', 'localconf')
38
39 test.addSubtest('10.10')
40 test.addSubtestTitle('10.10', "Begin the formation of an adjacency")
41 test.addSubtestDependency('10.10', 'localconf')
42 test.addSubtestDependency('10.10', 'routerconf')
43
44 test.addSubtest('ospfdisable', task = True)
45 test.addSubtestTitle('ospfdisable', "Disable OSPF on the router")
46 test.addSubtestDependency('ospfdisable', 'routerconf')
47
48 test.addSubtest('finallocalconf', task = True)
49 test.addSubtestTitle('finallocalconf', "Restore local configuration")
50 test.addSubtestDependency('finallocalconf', 'localconf')

```

Local GNU/Linux system is configured with an entry on the routing table in order to send packets directed to AllSPFRouters through the LOCAL_INTERFACE device. Moreover, as the behaviour of an OSPF daemon is emulated without actually running it, ICMP protocol-unreachable packets are dropped, using *iptables* [10], before being sent:

```

# Local machine configuration
try:
    test.begin('localconf')
    localcommand("ip addr flush dev %s" % LOCAL_INTERFACE)
    localcommand("ip addr add %s/%s dev %s" % (LOCAL_IP, LOCAL_MASK,
                                                LOCAL_INTERFACE))
    test.addResult('localconf', "Local IP Address", LOCAL_IP)
    test.addResult('localconf', "Local Netmask", "/" + LOCAL_MASK)
    localcommand("ip link set %s up" % LOCAL_INTERFACE)
    test.addResult('localconf', "Local Interface", LOCAL_INTERFACE)
61
    localcommand("ip route add 224.0.0.0/8 dev %s" % LOCAL_INTERFACE)
    # avoid protocol-unreachable messages from this host
    localcommand("iptables -A OUTPUT -p icmp -m icmp --icmp-type protocol-
                unreachable -j DROP")
66
    # scapy interface
    conf.iface = LOCAL_INTERFACE

```

```

71      # resync scapy with the local routing table
    conf.route.resync()

71    except:
    test.end('localconf', TEST_FAILED)
else:
    test.end('localconf', TEST_OK)

```

Then the router is configured for OSPF operation:

```

# Now configure the router
try:
    test.begin('routerconf')
    router = JuniperJ2320(SERIALDEVICE)
    router.setUsername(ROUTER_USERNAME)
    router.setPassword(ROUTER_PASSWORD)

    test.addResult('routerconf', "Serial Device", SERIALDEVICE)
    test.addResult('routerconf', "Router Username", ROUTER_USERNAME)
    test.addResult('routerconf', "Router Password", "***")

    router.gotoconfig()
    router.sendcommand("delete interfaces %s unit 0 family inet" %
                      ROUTER_INTERFACE)
    router.confpromptexpect()
    router.sendcommand("set interfaces %s unit 0 family inet address %s/%s" \
                      % (ROUTER_INTERFACE, ROUTER_IP, ROUTER_MASK))
    router.sendcommand("set interfaces %s enable" % (ROUTER_INTERFACE))

    # OSPF configuration
    router.confpromptexpect()
    router.sendcommand("set routing-options router-id %s" % ROUTER_IP)
    router.confpromptexpect(timeout=10)
    router.sendcommand("set protocols ospf area %s interface %s priority %s" \
                      % (OSPF_AREA, ROUTER_INTERFACE, ROUTER_PRIORITY))
    router.confpromptexpect(timeout=10)
    router.sendcommand("set protocols ospf area %s interface %s enable" \
                      % (OSPF_AREA, ROUTER_INTERFACE))
    router.confpromptexpect(timeout=10)
    router.sendcommand("set protocols ospf enable")

    # commit
    router.commit()

    test.addResult('routerconf', "Router Interface", ROUTER_INTERFACE)
    test.addResult('routerconf', "Router IP Address", ROUTER_IP)
    test.addResult('routerconf', "Router Netmask", "/" + ROUTER_MASK)
    test.addResult('routerconf', "Router OSPF Priority", ROUTER_PRIORITY)
    test.addResult('routerconf', "OSPF Area", OSPF_AREA)

116  except TestDependencyException:
    # The dependencies were not met
    test.end('routerconf', TEST_SKIPPED)
except Exception, err:
    # An error occurred
    print type(err), err
    traceback.print_tb(sys.exc_info()[2])
    test.end('routerconf', TEST_FAILED)
except:
    # Unexpected error
    raise
else:
    # The test succeeded
    test.end('routerconf', TEST_OK)

```

The main subtest starts by checking for the success of dependent subtests using the `begin()` method, printing a message using the `announce()` method, sniffing a router-generated OSPF Hello packet and finally asserting its correctness:

```

132  # Now begin the formation of an adjacency
#

```

```

#          |ROU|          |LOC|
#+-----+-----+
#          #             #
#          Down          Down
#          #             #
#          Hello (DR=0,seen=0)   #
#          #             #
#          Hello (DR=LOC,seen=ROU,...)  -->
#          #             #
#          ExStart      <-- D-D (Seq=x,l,M,Master)
#          #             #
#          #             #
#          #             #
try:
    test.begin('10.10')

    test.announce("Wait for an OSPF Hello from the router")
    # Wait for an OSPF Hello from the router
    #             Hello (DR=0,seen=0)   -->
    #             sniffedpackets = sniff(count=1, lfilter = lambda x: x.haslayer(OSPF_Hello),
    #                               timeout=60)
    assert len(sniffedpackets) > 0
    sniffedpackets.show()
    rp1 = sniffedpackets[0]
    pospf = rp1.getlayer(OSPF_Hdr)
    pospf.display()
    assert pospf.type == 1
    assert pospf.version == 2
    assert pospf.src == ROUTER_IP
    assert pospf.area == OSPF_AREA
    assert pospf.prio == ROUTER_PRIORITY
    assert pospf.authtype == 0

```

For a detailed description of Scapy functions and objects, please refer to [2]. In the following code fragment, an OSPF Hello packet with a priority value greater than the router's is forged, copying some fields from the afore-captured Hello packet and including the router-id in the *neighbor* field:

```

166     test.announce("Reply to the Hello including the router as neighbor")
# Reply to the Hello including the router as neighbor
#                               Hello (DR=LOC,seen=ROUTER_IP,...)
#
# p1 = IP() / OSPF_Hdr() / OSPF_Hello()
#
171     p1[IP].src = LOCAL_IP
p1[IP].dst = AllSPFRouters

176     p1[OSPF_Hdr].src = LOCAL_IP
p1[OSPF_Hdr].len = 48 # scapy_ospf bug

181     p1[OSPF_Hello].mask = LOCAL_FULL_MASK
p1[OSPF_Hello].options = 'E'
p1[OSPF_Hello].hellointerval = pospf.hellointerval
p1[OSPF_Hello].deadinterval = pospf.deadinterval
p1[OSPF_Hello].prio = LOCAL_PRIORITY
p1[OSPF_Hello].router = LOCAL_IP # DR
p1[OSPF_Hello].neighbor = ROUTER_IP # seen

186     send(p1)

```

Now a Database Description packet from the router is expected. Sniff it and check its correctness:

```
191     test.announce("Wait for a Database Description Packet")
# Wait for a Database Description
#                                     D-D (Seq=x, l, M, Master)
#                                     _____>
sniffedpackets = sniff(count = 1, lfilter = lambda x: x.haslayer(
    OSPF_DBDesc), timeout = 30)
assert len(sniffedpackets) > 0
sniffedpackets.show()
```

```

196     rp2 = sniffedpackets[0]
197     pospf = rp2.getlayer(OSPF_Hdr)
198     pospf.display()
199     assert pospf.type == 2
200     assert pospf.version == 2
201     assert pospf.src == ROUTER_IP
202     assert pospf.area == OSPF_AREA
203     assert pospf.authtype == 0
204     assert pospf.dbdescr == 7
205
206     test.announce("Correct Database Description Packet received")
207
208 except TestDependencyException:
209     # The dependencies were not met
210     test.end('10.10', TEST_SKIPPED)
211 except Exception, err:
212     # An error occurred
213     print type(err), err
214     traceback.print_tb(sys.exc_info() [2])
215     test.end('10.10', TEST_FAILED)
216 except:
217     # Unexpected error
218     raise
219 else:
220     # The test succeeded
221     test.end('10.10', TEST_OK)

```

The remainder of the test performs final configuration tasks, such as disabling OSPF on the router and flushing previously inserted firewall rules:

```

# now disable ospf on the router
try:
    test.begin('ospfdisable')
    router.confpromptexpect()
    router.sendcommand("set protocols ospf disable")
    router.commit()
    router.gotologinscreen()

except TestDependencyException:
    # The dependencies were not met
    test.end('ospfdisable', TEST_SKIPPED)
except Exception, err:
    # An error occurred
    print type(err), err
    traceback.print_tb(sys.exc_info() [2])
    test.end('ospfdisable', TEST_FAILED)
except:
    # Unexpected error
    raise
else:
    # The test succeeded
    test.end('ospfdisable', TEST_OK)

247
try:
    test.begin('finallocalconf')

252     localcommand("ip route del 224.0.0.0/8 dev %s" % LOCAL_INTERFACE)
     localcommand("iptables -D OUTPUT -p icmp -m icmp --icmp-type protocol-
      unreachable -j DROP")

except TestDependencyException:
    # The dependencies were not met
    test.end('finallocalconf', TEST_SKIPPED)
except Exception, err:
    # An error occurred
    print type(err), err
    traceback.print_tb(sys.exc_info() [2])
    test.end('finallocalconf', TEST_FAILED)
except:
    # Unexpected error

```

```

267     raise
else:
    # The test succeeded
    test.end('finallocalconf', TEST_OK)

```

Finally, the test is printed on the standard output and saved:

```

271     print test
test.save(dir = TEST_RUN_DIR)

```

Table 3 summarizes an execution of the test.

3.2.2 Adjacency Initial Forming Test on Cisco 2811

This section shows changes to listing 11 needed to adapt it to the execution with a connected Cisco 2811 router. The complete code can be found in appendix B.2.2.

The preamble is similar to the one of listing 11:

```

from scapy_ospf import *
from localconf import *
from cisco2811 import *
4 from testsummary import *
import sys
import traceback

# Verify that the router behaves as in section 10.10 of RFC 2328 (OSPFv2).
9 # where an adjacency forming example is shown

AllSPFRouters = '224.0.0.5'

SERIALDEVICE = '/dev/ttyUSB0'
14 ROUTER_IP = '191.168.0.31'
ROUTER_MASK = '255.255.255.0'
ROUTER_USERNAME = 'admin'
ROUTER_PASSWORD = 'secret'
ROUTER_HOSTNAME = 'cisco2'
19 ROUTER_INTERFACE = "FastEthernet 0/0"
LOCAL_INTERFACE = 'eth0'
LOCAL_IP = '191.168.0.32'
LOCAL_MASK = '24'
LOCAL_FULL_MASK = '255.255.255.0'
24 OSPF_AREA = '0.0.0.0'
ROUTER_PRIORITY = 100
LOCAL_PRIORITY = 200

TEST_RUN_DIR = "./test-runs"

```

In the Test object creation a suitable title string is given. Subtest-related definitions are omitted, as no changes are made.

```
test = Test("Cisco 2811 RFC2328 Section 10.10 Example Conformance")
```

Also the local configuration ('localconf') fragment is identical to the one in listing 11, but the subsequent router-dependent section ('routerconf') needs changes in the type of object being instantiated, related method calls and router command strings:

```

# Now configure the router
try:
    test.begin('routerconf')
    router = Cisco2811(SERIALDEVICE)
82    router.setUsername(ROUTER_USERNAME)
    router.setPassword(ROUTER_PASSWORD)
    router.setHostname(ROUTER_HOSTNAME)

    test.addResult('routerconf', "Serial Device", SERIALDEVICE)
87    test.addResult('routerconf', "Router Username", ROUTER_USERNAME)

```

```

    test.addResult('routerconf', "Router Password", "***")
    test.addResult('routerconf', "Router Hostname", ROUTER_HOSTNAME)

92     router.gotoconfig()
93     router.sendcommand("interface %s" % ROUTER_INTERFACE)
94     router.config_promptexpect("if")
95     router.sendcommand("ip address %s %s" % (ROUTER_IP, ROUTER_MASK))
96     router.config_promptexpect("if")
97     router.sendcommand("no shutdown")
98     router.config_promptexpect("if")
99     router.sendcommand("end")
100    router.enabledpromptexpect()

102   # OSPF configuration
103   router.gotoconfig()
104   router.sendcommand("router ospf 100")
105   router.config_promptexpect("router")
106   router.sendcommand("network %s 255.255.255.255 area %s" %(ROUTER_IP,
107     OSPF_AREA))
108   router.config_promptexpect("router")
109   router.sendcommand("end")
110   router.enabledpromptexpect()

112   router.gotoconfig()
113   router.sendcommand("interface %s" % ROUTER_INTERFACE)
114   router.config_promptexpect("if")
115   router.sendcommand("ip ospf priority %s" % ROUTER_PRIORITY)
116   router.config_promptexpect("if")
117   router.sendcommand("end")
118   router.enabledpromptexpect()

120   test.addResult('routerconf', "Router Interface", ROUTER_INTERFACE)
121   test.addResult('routerconf', "Router IP Address", ROUTER_IP)
122   test.addResult('routerconf', "Router Netmask", "/" + ROUTER_MASK)
123   test.addResult('routerconf', "Router OSPF Priority", ROUTER_PRIORITY)
124   test.addResult('routerconf', "OSPF Area", OSPF_AREA)

127   except TestDependencyException:
128     # The dependencies were not met
129     test.end('routerconf', TEST_SKIPPED)
130   except Exception, err:
131     # An error occurred
132     print type(err), err
133     traceback.print_tb(sys.exc_info()[2])
134     test.end('routerconf', TEST_FAILED)
135   except:
136     # Unexpected error
137     raise
138   else:
139     # The test succeeded
140     test.end('routerconf', TEST_OK)

```

The central subtest, labeled '10.10', is leaved unchanged. In the remainder of the script, only OSPF disablement code needs to be adapted:

```

# now disable ospf on the router
233 try:
234   test.begin('ospfdisable')
235   router.gotoconfig()
236   router.sendcommand("no router ospf 100")
237   router.configpromptexpect()
238   router.gotologinscreen()

241   except TestDependencyException:
242     # The dependencies were not met
243     test.end('ospfdisable', TEST_SKIPPED)
244   except Exception, err:
245     # An error occurred
246     print type(err), err
247     traceback.print_tb(sys.exc_info()[2])
248     test.end('ospfdisable', TEST_FAILED)
249   except:

```

```

# Unexpected error
raise
else:
# The test succeeded
test.end('ospfdisable', TEST.OK)

```

253 An execution with a connected Cisco 2811 router gave as a result the test summary reported in table 4.

4 Conclusions and Further Work

In section 2 a new Python framework for router testing was introduced, while section 3 demonstrated its use in the performance of OSPF Version 2 tests on Juniper J2320 and Cisco 2811 routers.

The framework, which works in practice and permits the reuse of the tests on different routers at the cost of changes in the router configuration sections of the test scripts, may be enhanced and extended in several ways, some of which are described below.

Other Python libraries and language features may be combined with the framework too. In this way complex tests using threads, accessing the router's Web GUI, or implementing entire RFC sections, might be, for example, developed.

4.1 Defects

Sometimes, in order to achieve successful tests, several test runs have to be performed. The reason might be hardware (connection or transmission defects) or software (wrong timings).

Moreover, the code is not portable to non-POSIX operating systems, due to the use of the pexpect (§ 2.2) and serial.serialposix (§ 2.3) modules. It should be portable to other POSIX-compliant operating systems, but has been tested and executed only on a GNU/Linux system.

Finally, it should be pointed out that the efficiency of juniperj2320.JuniperJ2320 and cisco2811.Cisco2811 classes is suboptimal, but improvable, in terms of router-specific commands per method call.

4.2 Extensions

In a scenario where a same set of tests has to be performed on different router models, an object-oriented API for vendor-independent router configuration and communication could be defined. This could allow the reuse of the same, unchanged, test scripts for different router models.

The work done for NetML [11] could be adapted for this purpose, but instead of a transformation from a generic XML router configuration description to a router-specific configuration file, a transformation from a Python class describing a generic router configuration to commands given directly, e.g. via serial port, to the router could be performed.

Also a database containing router-dependent serial port configuration parameters (e.g. baud rates, stop bits, ...) could be included. By specifying (or even, if possible, autodiscovering) the router model, the associated serial port configuration parameters could be retrieved and the appropriate configuration algorithm could be

Juniper J2320 RFC2328 Section 10.10 Example Conformance	
Local setup	DONE
Local IP Address	191.168.0.32
Local Netmask	/24
Local Interface	eth0
Router setup	DONE
Serial Device	/dev/ttyUSB0
Router Username	root
Router Password	***
Router Interface	ge-0/0/0
Router IP Address	191.168.0.31
Router Netmask	/24
Router OSPF Priority	100
OSPF Area	0.0.0.0
Begin the formation of an adjacency	PASSED
Disable OSPF on the router	DONE
Restore local configuration	DONE

Table 3: The summary of the Adjacency Initial Forming Test performed on a Juniper J2320 router.

Cisco 2811 RFC2328 Section 10.10 Partial Example Conformance	
Local setup	DONE
Local IP Address	191.168.0.32
Local Netmask	/24
Local Interface	eth0
Router setup	DONE
Serial Device	/dev/ttyUSB0
Router Username	admin
Router Password	***
Router Hostname	cisco2
Router Interface	FastEthernet 0/0
Router IP Address	191.168.0.31
Router Netmask	/255.255.255.0
Router OSPF Priority	100
OSPF Area	0.0.0.0
Begin the formation of an adjacency	PASSED
Disable OSPF on the router	DONE
Restore local configuration	DONE

Table 4: The summary of the Adjacency Initial Forming Test performed on a Cisco 2811 router.

selected. Router error messages should be considered as well, perhaps by raising appropriate exceptions at the moment of their appearance.

A simple script using such interface could be similar to the following:

```

2   router = Router(JUNIPER_J2320)
3   router.setUsername('root')
4   router.setPassword('secret')
5   router.setConnectionMethod(SERIAL_CONNECTION) # The router is connected to the
6   # machine running this script
7   # through a serial port
8
9   router.setSerialport('/dev/ttyUSB0')           # The serial port device
10  router.setIPv4Address(0, '192.168.2.1')       # Interface no. and IP address
11
12  router.send()                                # Sync the router's configuration
13  # with the Router object
14
15  # Perform some tests using packet forging
16  # ...
17  # ...
18
19  router.retrieve()                            # Retrieve the state of the router
20
21  router.showOSPFDatabase()

```

Furthermore, the API could be extended to allow not only router configuration via serial port, but also, where possible, using the SSH or Telnet protocols⁶. This feature could be exploited to configure an entire network from a single terminal, allowing the implementation of complex tests involving several routers.

Moreover, once a configuration has been defined on router objects, these could participate in a network simulation, in order to, e.g., evaluate the validity of a network configuration before actually committing changes to “real” routers.

The testsummary module could be enhanced as well, to allow subtest code reuse and lighten the syntax. Encapsulating the code between the `try: ... except:` statements in an overridable method of the `testsummary.SubTest` class, and using a more object-oriented style for the definition of subtests and their interdependencies, could achieve this goal.

For example, `run()` and `execute()` methods similar to the following could be added to the `SubTest` class definition that can be found in listing 7:

```

class SubTest():
    #
    #
3    def execute(self):
        "This method should be overridden by derived classes."
        pass

8    def run(self):
        "Encapsulates the call to execute()"
        try:
            self.checkForDependencies() # method to be defined too
            self.execute()
13    except TestDependencyException:
        # The dependencies were not met
        self.setFinalResult(TEST_SKIPPED)
    except Exception, err:
        # An error occurred
        print type(err), err
        traceback.print_tb(sys.exc_info()[2])
        self.setFinalResult(TEST_FAILED)
    except:
        # Unexpected error
        raise
23

```

⁶Note that Python already includes SSH and Telnet libraries in its standard distribution.

```

else:
    # The test succeeded
    self.setFinalResult(TEST_OK)

```

Then a more specific class could be derived from SubTest:

```

3 class LocalConfSubTest(SubTest):
        def execute(self):
            localcommand("ip addr flush dev eth0")
            localcommand("ip addr add 192.168.0.1/24 dev eth0")
            localcommand("ip link set eth0 up")

```

But the drawback could be less readable test scripts, due to code spreading across different objects or files.

The test summaries could be enhanced as well, by showing the execution time of the various subtests, including a pdf output and improving the \TeX output.

References

- [1] Python Software Foundation. Python programming language – official website. <http://www.python.org/>, 2008.
- [2] Philippe Biondi. Scapy. <http://www.secdev.org/projects/scapy/>, 2008.
- [3] Noah Spurrier et al. Pexpect - noah.org. <http://www.noah.org/wiki/Pexpect>, 2008.
- [4] Chris Liechti. pySerial. <http://pyserial.wiki.sourceforge.net/pySerial>, 2008.
- [5] J. Moy. OSPF version 2. IETF RFC 2328, April 1998.
- [6] Andrew S. Tanenbaum, Paolo Canali, and Alessandro Valli. *Reti di calcolatori 4a ed.* Pearson Education Italia, 2003.
- [7] Dirk Loss. OSPF extension for scapy. <http://trac.secdev.org/scapy/wiki/OSPF>, 2008.
- [8] POSIX. Standards. IEEE.
- [9] Iproute2. <http://www.linuxfoundation.org/en/Net:Iproute2>, 2008.
- [10] Iptables. <http://www.netfilter.org/>, 2008.
- [11] <NetML/>. <http://www.dia.uniroma3.it/~compunet/netml/index.html>, 2008.
- [12] Free Software Foundation, Inc. The GNU general public license. <http://www.gnu.org/licenses/gpl.html>, June 2007.

A Source Code of Some Framework Components

This appendix reports only the components developed ad-hoc for the completion of the framework described in section 2. The other components' source code can be found in [2, 7, 3, 4].

A.1 The serialrouter module

Listing 4: The serialrouter module.

```

# Copyright (C) Claudio Pisa 2008
## clauz at ninux.org
## You are free to use and modify this code according
4 ## to the GNU Public Licence version 3 and subsequent versions.
## Visit www.gnu.org for details.

import serial
import fdpexpect
import time

DEFAULTTIMEOUT = 8
SLEEPAFTERWRITE = 2
COMMANDSLOWNESS = 0.05

14 class RouterConfigurationException(Exception):
    "Error in the router configuration process."
    pass

19 class SerialConnectedRouter(serial.Serial, fdpexpect.fdpawn):
    """
        This class represents a router connected via serial port to the
        machine running this program.
        This class is POSIX specific, due to the fdpexpect module, which in its
        turn uses the POSIX specific pty module and to the Serial.fileno() call
        (see below).
    """
    def __init__(self, serialdevice='/dev/ttyS0', baudrate=9600, \
                 bytesize=8, parity='N', stopbits=1, timeout=DEFAULTTIMEOUT):
        """
            For the serialdevice, baudrate, bytesize, parity, stopbits and
            timeout parameters please refer to serial.Serial documentation.

        """
        # Open the serial device
        serial.Serial.__init__(self, serialdevice, baudrate=baudrate, \
                              bytesize=bytesize, parity=parity, stopbits=stopbits, \
                              timeout=timeout)
        time.sleep(2)

39     # Attach an expect/spawn instance to
     # the serial device (POSIX specific call)
     fdpexpect.fdpawn.__init__(self, self.fileno())

44     # Turn off buffering
     self.maxread=1

def __del__(self):
    "Destroyer. Close the serial port."
49     self.close()
     fdpexpect.fdpawn.__del__(self)

def sendline(self, line):
    "Send a line to the router. Overrides fdpexpect.spawn.sendline()"
54     self.send(line + "\r\n")

def sendcommand(self, command):
    """Send a command to the router, character by character.
    Some routers (e.g. Juniper J2320) don't need this,
    i.e. the sendline() method is enough for them."""
59     print "Sending: %s" % command
     for c in command:
         self.send(c)
         time.sleep(COMMANDSLOWNESS)
     self.sendline("")
     time.sleep(SLEEPAFTERWRITE)
     self.flush()
     self.flushOutput()

64     def timedexpect(self, pattern, timeout = DEFAULTTIMEOUT, quiet = False):

```

```

    "Like spawn.expect(), but raise an exception on timeout"
    if not quiet:
        print "Expecting: %s" % pattern
    res = self.expect([pattern, fdpexpect.TIMEOUT], timeout=timeout)
    if res == 1:
        raise RouterConfigurationException("Timeout occurred.")

74 def listexpect(self, patterns, timeout = DEFAULTTIMEOUT, quiet = False):
75     res = self.expect(patterns, timeout = timeout)
76     self.flushInput()
77     if not quiet:
78         print "Matched %s" % patterns[res]
79     return res

84 def readuntil(self, pattern, timeout = DEFAULTTIMEOUT):
85     "Reads from the serial device until a line matching pattern is met"
86     self.expect(pattern, timeout)
87     return self.before

```

A.2 The juniperj2320 module

Listing 5: The juniperj2320 module.

```

1 # Copyright (C) Claudio Pisa 2008
# clauz at ninux.org
# You are free to use and modify this code according
# to the GNU Public Licence version 3 and subsequent versions.
# Visit www.gnu.org for details.

6 from serialrouter import *
import fdpexpect
import time

11 TIMEOUT = 4
BAUDRATE = 9600
BYTESIZE = 8
PARITY = 'N'
STOPBITS = 1
16 TIMEOUT_ERROR = "Timeout occurred."
EOF_ERROR = "Possible serial communication error. Please check that no other
            program is accessing the serial port."

class JuniperJ2320(SerialConnectedRouter):
    """
21     This class represents a Juniper J2320 Router connected via
     serial port.
    """
26     def __init__(self, serialdevice):
27         self.loginprompt = "login:"
28         self.initprompt = "%%"
29         self.cliprompt = "\>"
30         self.confprompt = "\#"
31         self.username = None
32         self.password = None
33         SerialConnectedRouter.__init__(self, serialdevice, BAUDRATE, \
34                                         BYTESIZE, PARITY, STOPBITS, TIMEOUT)

36     def setUsername(self, username):
37         self.username = username
38         self.initprompt = "%s@.*%%" % self.username
39         self.cliprompt = "%s@.*\>" % self.username
40         self.confprompt = "%s@.*\#" % self.username
41         self.promptlist = [self.loginprompt, self.initprompt,
42                           self.cliprompt, self.confprompt,
43                           fdpexpect.TIMEOUT, fdpexpect.EOF]

44     def setPassword(self, password):
45         self.password = password

46     def gotologinscreen(self, sendnewline = True):
47         """

```

```

    Climb the JunOS configuration hierarchy
    until the login prompt appears
    """
51     if sendnewline:
52         self.sendline("")
53     i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
54     if i == 0: # loginprompt
55         pass # success
56     # initprompt, cliprompt, confprompt
57     elif i == 1 or i == 2 or i == 3:
58         self.sendcommand("exit")
59         self.gotologinscreen(sendnewline=False)
60     elif i == 4: # timeout
61         raise RouterConfigurationException(TIMEOUT_ERROR)
62     elif i == 5: # eof
63         raise RouterConfigurationException(EOF_ERROR)

64     def login(self):
65         if self.username == None or self.password == None:
66             raise RouterConfigurationException("Username or password not set.")
67         self.sendline("")
68         i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
69         if i != 0:
70             self.gotologinscreen()
71             self.sendcommand(self.username)
72             self.timedexpect("ssword:", timeout = 12)
73             self.sendcommand(self.password)
74             self.initpromptexpect(timeout = 20)

75     def gotocli(self, sendnewline = True):
76         if sendnewline:
77             self.sendline("")
78         i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
79         if i == 0: # loginprompt
80             self.login()
81             self.gotocli(sendnewline = False)
82             return
83         elif i == 1: # initprompt
84             self.sendcommand("cli")
85             self.gotocli(sendnewline = False)
86             return
87         elif i == 2: # cliprompt
88             pass # success
89         elif i == 3: # confprompt
90             self.sendcommand("exit")
91             self.gotocli(sendnewline = False)
92             return
93         elif i == 4: # timeout
94             raise RouterConfigurationException(TIMEOUT_ERROR)
95         elif i == 5: # eof
96             raise RouterConfigurationException(EOF_ERROR)
97         self.sendcommand("set cli screen-length 0")
98         self.timedexpect("length set to 0")
99         self.clipromptexpect()

100     def gotoconf(self, sendnewline = True):
101         if sendnewline:
102             self.sendline("")
103         i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
104         if i == 0: # loginprompt
105             self.login()
106             self.gotoconf(sendnewline = True)
107         elif i == 1: # initprompt
108             self.gotocli()
109             self.gotoconf(sendnewline = True)
110         elif i == 2: # cliprompt
111             self.sendcommand("configure")
112             self.gotoconf(sendnewline = False)
113         elif i == 3: # confprompt
114             pass # success
115         elif i == 4: #timeout
116             raise RouterConfigurationException(TIMEOUT_ERROR)
117         elif i == 5:

```

```

121         raise RouterConfigurationException(EOF_ERROR)

122     def commit(self):
123         self.confpromptexpect()
124         self.sendcommand("commit")
125         self.timedexpect("commit complete", timeout=20)

126     def confpromptexpect(self, timeout=DEFAULTTIMEOUT):
127         try:
128             self.timedexpect(self.confprompt, timeout)
129         except RouterConfigurationException:
130             self.sendline("")
131             self.timedexpect(self.confprompt, timeout)

132     def clipromptexpect(self, timeout=DEFAULTTIMEOUT):
133         try:
134             self.timedexpect(self.cliprompt, timeout)
135         except RouterConfigurationException:
136             self.sendline("")
137             self.timedexpect(self.cliprompt, timeout)

138     def initpromptexpect(self, timeout=DEFAULTTIMEOUT):
139         try:
140             self.timedexpect(self.initprompt, timeout)
141         except RouterConfigurationException:
142             self.sendline("")
143             self.timedexpect(self.initprompt, timeout)

```

A.3 The cisco2811 module

Listing 6: The cisco2811 module.

```

1 # Copyright (C) Claudio Pisa 2008
# clauz at ninux.org
# You are free to use and modify this code according
# to the GNU Public Licence version 3 and subsequent versions.
# Visit www.gnu.org for details.

6   from serialrouter import *
import fdpexpect
import time

11  TIMEOUT = 4
BAUDRATE = 115200
BYTESIZE = 8
PARITY = 'N'
STOPBITS = 1
TIMEOUT.ERROR = "Timeout occurred."
EOF_ERROR = "Possible serial communication error. Please check that no other
program is accessing the serial port."
ADDITIONALSLEEP = 4

21  class Cisco2811(SerialConnectedRouter):
    """
    This class represent a Cisco 2811 Router connected via
    serial port.
    """
26  def __init__(self, serialdevice):
        self.loginprompt = "sername:"
        self.passwordprompt = "ssword:"
        self.cliprompt = '>'
        self.enabledprompt = r"[^()]*"
        self.configprompt = r".config.*"
        self.config_prompt = r".config-.*"
        self.username = None
        self.password = None
        self.hostname = None
        self.tries = 0
SerialConnectedRouter.__init__(self, serialdevice, BAUDRATE, \
                                BYTESIZE, PARITY, STOPBITS, TIMEOUT)

```

```

41     def setUsername(self, username):
42         self.username = username
43
44     def setPassword(self, password):
45         self.password = password
46
47     def setHostname(self, hostname):
48         self.hostname = hostname
49         self.enabledprompt = "%s#" % hostname
50         self.cliprompt = "%s>" % hostname
51         self.configprompt = "%s.config.#" % hostname
52         self.config_prompt = "%s.config-*#" % hostname
53         self.promptlist = [self.loginprompt, self.cliprompt, self.
54                           enabledprompt, \
55                           self.passwordprompt, self.configprompt, self.config_prompt, \
56                           fdexpect.TIMEOUT, fdexpect.EOF]
57
58     def gotologinscreen(self):
59         """
60             Climb the IOS configuration hierarchy
61             until the login prompt appears
62         """
63         self.sendcommand("")
64         i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
65         if i == 0: # loginprompt
66             pass # success
67             # cliprompt, enabledprompt, configprompt, config_prompt
68         elif i == 1 or i == 2 or i == 4 or i == 5:
69             self.sendcommand("exit")
70             time.sleep(ADDITIONALSLEEP)
71             self.gotologinscreen()
72         elif i == 3: # passwordprompt
73             self.sendcommand("")
74             self.gotologinscreen()
75         elif i == 6: #timeout
76             if self.tries == 0:
77                 self.sendcommand("")
78                 self.tries += 1
79             else:
80                 self.tries = 0
81                 raise RouterConfigurationException(TIMEOUT_ERROR)
82         elif i == 7:
83             raise RouterConfigurationException(EOF_ERROR)
84
85     def login(self):
86         if self.username == None or self.password == None or self.hostname == None:
87             raise RouterConfigurationException("Username or password or
88                                             hostname not set.")
89         time.sleep(ADDITIONALSLEEP)
90         i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
91         if i != 0:
92             self.gotologinscreen()
93             self.sendcommand("")
94             self.login()
95             return
96         self.timedexpect(self.loginprompt)
97         self.sendcommand(self.username)
98         self.timedexpect(self.passwordprompt, timeout = 12)
99         self.sendcommand(self.password)
100        self.clipromptexpect(timeout = 20)
101        self.sendcommand("terminal length 0")
102        self.clipromptexpect()
103
104    def gotocli(self, sendnewline = True):
105        if sendnewline:
106            self.sendcommand("")
107        i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
108        if i == 0: # loginprompt
109            self.login()
110            self.gotocli()
111            return
112        elif i == 1: # cliprompt: bingo
113            pass

```

```

111     elif i == 2 or i == 4 or i == 5: # enabledprompt, configprompt,
112         config_prompt
113             self.sendcommand("exit")
114             time.sleep(ADDITIONALSLEEP)
115             self.gotocli(sendnewline = True)
116             return
117         elif i == 3: # passwordprompt
118             self.sendcommand("")
119             self.gotocli(sendnewline = False)
120             return
121         elif i == 6: # timeout
122             raise RouterConfigurationException(TIMEOUT_ERROR)
123         elif i == 7:
124             raise RouterConfigurationException(EOF_ERROR)
125             self.sendcommand("terminal length 0")
126             self.clipromptexpect()
127
128     def gotoenabled(self, sendnewline = True):
129         if sendnewline:
130             self.sendcommand("")
131             i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
132             if i == 0: # loginprompt
133                 self.login()
134                 self.gotoenabled()
135             elif i == 1: # cliprompt
136                 self.sendcommand("enable")
137                 time.sleep(ADDITIONALSLEEP)
138                 self.gotoenabled(sendnewline = False)
139             elif i == 2: # enabledprompt: bingo
140                 pass
141             elif i == 3: # passwordprompt
142                 self.sendcommand(self.password)
143                 self.gotoenabled(sendnewline = False)
144             elif i == 4 or i == 5: # configprompt, config_prompt
145                 self.sendcommand("exit")
146                 time.sleep(ADDITIONALSLEEP)
147                 self.gotoenabled(sendnewline = True)
148             elif i == 6: # timeout
149                 raise RouterConfigurationException(TIMEOUT_ERROR)
150             elif i == 7:
151                 raise RouterConfigurationException(EOF_ERROR)
152
153     def gotoconfig(self, sendnewline = True):
154         if sendnewline:
155             self.sendcommand("")
156             i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
157             if i == 0: # loginprompt
158                 self.login()
159                 self.gotoenabled()
160                 self.gotoconfig()
161             elif i == 1: # cliprompt
162                 self.gotoenabled()
163                 self.gotoconfig()
164             elif i == 2: # enabledprompt
165                 self.sendcommand("configure terminal")
166                 self.gotoconfig(sendnewline = False)
167             elif i == 3: # passwordprompt
168                 self.sendcommand("")
169                 self.gotoconfig(sendnewline = False)
170             elif i == 4: # configprompt: bingo
171                 pass
172             elif i == 5: # config_prompt
173                 self.sendcommand("exit")
174                 time.sleep(ADDITIONALSLEEP)
175                 self.gotoconfig(sendnewline = True)
176             elif i == 6: # timeout
177                 raise RouterConfigurationException(TIMEOUT_ERROR)
178             elif i == 7:
179                 raise RouterConfigurationException(EOF_ERROR)
180
181     def write(self):
182         self.gotoenabled()
183         self.sendcommand("write")
184         self.timedexpect("[OK]", timeout=20)

```

```

186     def enabledpromptexpect(self, timeout=DEFAULTTIMEOUT):
187         try:
188             self.timedexpect(self.enabledprompt, timeout)
189         except RouterConfigurationException:
190             self.sendcommand("")
191             self.timedexpect(self.enabledprompt, timeout)
192
193     def configpromptexpect(self, timeout=DEFAULTTIMEOUT):
194         try:
195             self.timedexpect(self.configprompt, timeout)
196         except RouterConfigurationException:
197             self.sendcommand("")
198             self.timedexpect(self.configprompt, timeout)
199
200     def config_promptexpect(self, subprompt=".*", timeout=DEFAULTTIMEOUT):
201         config_p = "%s.config-%s.#" % (self.hostname, subprompt)
202         try:
203             self.timedexpect(config_p, timeout)
204         except RouterConfigurationException:
205             self.sendcommand("")
206             self.timedexpect(config_p, timeout)
207
208     def clipromptexpect(self, timeout=DEFAULTTIMEOUT):
209         try:
210             self.timedexpect(self.cliprompt, timeout)
211         except RouterConfigurationException:
212             self.sendcommand("")
213             self.timedexpect(self.cliprompt, timeout)

```

A.4 The testsummary module

Listing 7: The testsummary module.

```

# Copyright (C) Claudio Pisa 2008
# clauz at ninux.org
# You are free to use and modify this code according
# to the GNU Public Licence version 3 and subsequent versions.
# Visit www.gnu.org for details.

"""
Run and summarize tests.
Example:

12      # A new Test() instance
13      test = Test("Foo Bar")

14      # Initial setup
15      test.addSubtest('SetUp', task=True)
16      test.addSubtestTitle('SetUp', "Initial configuration")
17      try:
18          test.begin('SetUp')
19          # Perform set-up operations
20          # ...
21      except:
22          test.end('SetUp', TEST_FAILED)
23      else:
24          # The setup succeeded
25          test.end('SetUp', TEST_OK)

26
27      # First sub-test
28      test.addSubtest('FirstTest') # labels may be strings or numbers
29      test.addSubtestTitle('FirstTest', "Just a test")
30      test.addSubtestDependency('FirstTest', 'SetUp') # 'FirstTest' depends on '
31          'SetUp'
32      # begin the test
33      try:
34          test.begin('FirstTest')
35          # Perform the sub-test

```

```

37     # ...
42     # Do some assertions
43     # assert(...)
44     # ...
45     # Collect some results during the sub-test
46     test.addResult('FirstTest', 'HelloInterval', 10)
47     # ...
48     test.addResult('FirstTest', 'DeadInterval', 40)
49     # ...
50     test.addResult('FirstTest', 'Flags', 'E+M')
51 except TestDependencyException:
52     # The dependencies were not met
53     test.end('FirstTest', TEST_SKIPPED)
54 except Exception, err:
55     # An error occurred
56     print type(err), err
57     test.end('FirstTest', TEST_FAILED)
58 except:
59     # Unexpected error
60     raise
61 else:
62     # The test succeeded
63     test.end('FirstTest', TEST_OK)

# Second sub-test
64 test.addSubtest(2)
65 test.addSubtestTitle(2, "Another test")
66 # begin the test
67 try:
68     test.begin(2)
69     # Perform the sub-test
70     # ...
71 except Exception, err:
72     # An error occurred
73     print type(err), err
74     test.end(2, TEST_FAILED)
75 except:
76     # Unexpected error
77     raise
78 else:
79     # The test succeeded
80     test.end(2, TEST_OK)

# Output the summary on the screen
81 print test
82
83 # Save the test summary
84 test.save()

"""
85
86 import os

87 TEXTWIDTH = 80
88 DESCFIELD = 0.6 # percentage of TEXTWIDTH
89 SEPFIELD = 0.1 # percentage of TEXTWIDTH
90 VALFIELD = 0.3 # percentage of TEXTWIDTH

91 TEST_FAILED = 0
92 TEST_OK = 1
93 TEST_SKIPPED = 2

# Find ansi color codes
94 try:
95     import curses
96 except:
97     colorcodes = {
98         "black": "\033[30m",
99         "red": "\033[31m",
100        "green": "\033[32m",
101        "yellow": "\033[33m",
102        "blue": "\033[34m",
103        "magenta": "\033[35m",
104        "cyan": "\033[36m",
105    }

```

```

112         "white": ""
    }
else:
    curses.setupterm()
    __ansifg = curses.tigetstr('setaf')
    colorcodes = {
        "black": curses.tparm(__ansifg, 0),
        "red": curses.tparm(__ansifg, 1),
        "green": curses.tparm(__ansifg, 2),
        "yellow": curses.tparm(__ansifg, 3),
        "blue": curses.tparm(__ansifg, 4),
        "magenta": curses.tparm(__ansifg, 5),
        "cyan": curses.tparm(__ansifg, 6),
        "white": curses.tparm(__ansifg, 7)
    }

127 import pickle
import time

class TestException(Exception):
    pass

132 class TestDependencyException(TestException):
    pass

137 class InvalidCommandException(TestException):
    pass

142 class Result():
    """
    This class represents a result of a subtest
    """
    def __init__(self, description = "", value = None):
        "Initializes a Result instance"
        self.description = description
        self.value = value

    def setValue(self, value):
        "Set the value of the Result object"
        self.value = value

152    def setDescription(self, description):
        "Set the description of the Result object"
        self.description = description

    def getValue(self):
        "Get the value of the Result object"
        return self.value

    def getDescription(self):
        "Get the description of the Result object"
        return self.description

    def getDV(self):
        "Get the value and the description of the Result object"
        return (self.description, self.value)

    def __str__(self):
        descwidth = int(TEXTWIDTH * DESCFIELD)
        sepwidth = int(TEXTWIDTH * SEPFIELD)
        valwidth = int(TEXTWIDTH * VALFIELD)
        res = ""
        if isinstance(self.value, bool):
            if self.value:
                val = "Passed"
177        else:
                val = "Failed"
        else:
            val = self.value
        res += "%*s" % (descwidth, self.description)
        res += " " * sepwidth
        res += "%-*s" % (valwidth, val)
        res += "\n"

```

```

        return res

187 def getTeX(self):
    """Returns a string with the TeX representation of this object."""
    res = ""
    if isinstance(self.value, bool):
        if self.value:
            val = "Passed"
        else:
            val = "Failed"
    else:
        val = self.value
    res += "\t%s & %s \\\\\n" % (self.description, val)
    return res

202 class SubTest():
    """
    This class represents a subtest. Each subtest is made of
    various results.
    """
207     def __init__(self, title, task = False):
        """
        Initializes a SubTest instance.
        title is the title of the SubTest.
        task specifies if the current subtest represents
        a task (e.g. initial setup).
        """
        self.title = title
        self.results = list()
        self.finalresult = TEST_FAILED
        self.istask = task

217     def addResult(self, description, result):
        """
        Add a result to the subtest
        newresult = Result(description, result)
        self.results.append(newresult)

222     def setFinalResult(self, finalresult):
        """
        Specify if the entire subtest was passed (TEST_OK) or
        if it failed (TEST_FAILED), or skipped (TEST_SKIPPED).
        """
        self.finalresult = finalresult

227     def getFinalResult(self):
        """
        If the entire subtest was passed (TEST_OK) or
        if it failed (TEST_FAILED), or skipped (TEST_SKIPPED).
        """
        return self.finalresult

232     def getFinalResultString(self):
        """
        Returns a string representation of the final result
        of the subtest (e.g. "PASSED" or "FAILED" or "DONE")
        """
242         if self.istask:
            if self.finalresult == TEST_OK:
                return "DONE"
            elif self.finalresult == TEST_SKIPPED:
                return "SKIPPED"
            else:
                return "ERROR"
        else:
            if self.finalresult == TEST_OK:
                return "PASSED"
            elif self.finalresult == TEST_SKIPPED:
                return "SKIPPED"
            else:
                return "FAILED"

247     def getFinalResultColorString(self):

```

```

    """
    Like getFinalResultString but using ANSI colors.
    """
262 res = ""
263 if self.finalresult == TEST_OK:
264     res += colorcodes["green"]
265 else:
266     res += colorcodes["red"]
267 res += self.getFinalResultString()
268 res += colorcodes["white"]
269 return res

270 def setTitle(self, title):
271     "Set the title of the sub-test"
272     self.title = title

273 def getTitle(self):
274     "Get the title of the sub-test"
275     return self.title

276 def __str__(self):
277     descwidth = int(TEXTWIDTH * DESCFIELD)
278     sepwidth = int(TEXTWIDTH * SEPFIELD)
279     valwidth = int(TEXTWIDTH * VALFIELD)
280     res = ""
281     res += colorcodes["white"]
282     res += "=" * TEXTWIDTH
283     res += "\n"
284     res += colorcodes["yellow"]
285     res += "%*s" % (descwidth, self.title)
286     res += colorcodes["white"]
287     res += " " * (sepwidth)
288     res += "%-*s" % (valwidth, self.getFinalResultColorString())
289     res += "\n"

290     if self.results:
291         res += "-" * TEXTWIDTH
292         res += "\n"
293
294     for result in self.results:
295         res += str(result)

296
297     return res

298
299 def getTeX(self):
300     "Returns a string with the TeX representation of this object."
301     val = self.getFinalResultString()
302     res = ""
303     res += "\t\\hline \n"
304     res += "\t\\textbf{\%s} & \\textbf{\%s} \\\\\n" % (self.title, val)

305     if self.results:
306         res += "\t\\hline \n"
307
308     for result in self.results:
309         res += result.getTeX()

310
311     return res

312
313
314 class Test():
315     """
316     This class represents a test. Each test is made of various
317     subtests. Each subtest is made of various results.
318     """
319     def __init__(self, test_title):
320         """
321         Initializes a Test instance.
322         test_title should be a string with the title of the test.
323         """
324         self.title = test_title
325         self.subtests = dict()

```

```

        self.currentindex = 0
        self.subtestorder = dict()
        self.dependencies = dict()

337     def getTitle(self):
342         """Returns the title of the test."""
343         return self.title

347     def addSubtest(self, subtestlabel, task = False):
348         """
349             Add a subtest with label subtestlabel, which can be of
350             any immutable type (int, string, ...)
351             If task == True, the subtest is considered a task,
352             i.e. prints "DONE" or "ERROR" instead of "PASSED"
353             or "FAILED"
354         """
355         # We have a dictionary label -> subtest object
356         # and a dictionary index -> label
357         # to preserve the order of the subtests
358
359         newsubtest = SubTest(subtestlabel, task)
360         self.subtests.update({subtestlabel: newsubtest})
361         self.subtestorder.update({self.currentindex: subtestlabel})
362         self.currentindex += 1

363     def addSubtestDependency(self, subtestlabel, dependsonlabel):
364         """
365             Specifies that the subtest specified by subtestlabel depends
366             on the success of the subtest specified by dependsonlabel.
367         """
368         if not self.subtests.has_key(subtestlabel):
369             errstr = "Invoke addSubTest() first"
370             raise InvalidCommandException, errstr
371         testdeps = self.dependencies.get(subtestlabel, [])
372         testdeps.append(dependsonlabel)
373         self.dependencies.update({subtestlabel: testdeps})

374     def addSubtestTitle(self, subtestlabel, subtesttitle):
375         """
376             Give a title to the subtest
377         """
378         if not self.subtests.has_key(subtestlabel):
379             self.addSubtest(subtestlabel)
380         self.subtests[subtestlabel].setTitle(subtesttitle)

381     def printTitleString(self, outstring):
382         """
383             Used to print fancy messages on the screen
384         """
385         print colorcodes["blue"], "-" * TEXTWIDTH
386         print "\t", outstring
387         print colorcodes["blue"], "-" * TEXTWIDTH, colorcodes["white"]

388     def announce(self, outstring):
389         """
390             Used to print short messages on the screen
391         """
392         print ""
393         print colorcodes["yellow"], "\t", outstring, colorcodes["white"]
394         print ""

395     def begin(self, subtestlabel, failure_result_value = TEST_FAILED):
396         """
397             Begin the sub-test checking for subtest interdependencies.
398             failure_result_value is the value assigned to the subtest
399             in case of failure
400         """
401         if not self.subtests.has_key(subtestlabel):
402             self.addSubtest(subtestlabel)

403             self.subtests[subtestlabel].setFinalResult(failure_result_value)

404             # get dependencies
405             testdeps = self.dependencies.get(subtestlabel, [])

406             # check dependencies
407             for dep in testdeps:
408                 try:
409                     dept = self.subtests[dep]
410                 except KeyError:

```

```

407         errstr = "Test %s not found" % dep
408         raise InvalidCommandException, errstr
409
410     # if the test on which the current test depends
411     # were not passed, raise an exception
412     if dept.getFinalResult() != TEST_OK:
413         outstring = "%sSkip: %s%s.%s" \
414             % (colorcodes["yellow"], \
415                 colorcodes["magenta"], \
416                     self.subtests[subtestlabel].getTitle(), \
417                         colorcodes["white"])
418         self.printTitleString(outstring)
419         errstr = "Dependencies not met for test %s." % subtestlabel
420         raise TestDependencyException, errstr
421
422     outstring = "%sBegin: %s%s.%s" % (colorcodes["yellow"], \
423         colorcodes["magenta"], \
424             self.subtests[subtestlabel].getTitle(), \
425                 colorcodes["white"])
426     self.printTitleString(outstring)
427
428     def addResult(self, subtestlabel, description, result):
429         "Add a result to a sub-test"
430         self.subtests[subtestlabel].addResult(description, result)
431
432     def end(self, subtestlabel, finalresult = False):
433         """
434             Finalize the subtest, specifying the final result
435             (e.g. TEST_OK or TEST_FAILED)
436         """
437         try:
438             self.subtests[subtestlabel].setFinalResult(finalresult)
439         except KeyError:
440             errstr = "Test %s not found" % subtestlabel
441             raise InvalidCommandException, errstr
442
443         if finalresult == TEST_SKIPPED:
444             return
445         elif finalresult == TEST_OK:
446             resstr = colorcodes["green"]
447         else:
448             resstr = colorcodes["red"]
449
450
451         resstr += self.subtests[subtestlabel].getFinalResultString()
452         resstr += colorcodes["white"]
453         outstring = "%sFinish: %s%s.%s Result: %s%s" % (colorcodes["yellow"],
454             \
455                 colorcodes["magenta"], \
456                     self.subtests[subtestlabel].getTitle(), \
457                         colorcodes["white"], \
458                             resstr, \
459                                 colorcodes["white"])
460         self.printTitleString(outstring)
461
462     def __str__(self):
463         "Returns the summary of the test"
464         title = ""
465         title += " " + colorcodes["white"]
466         title += self.title
467         title += " " + colorcodes["blue"]
468         res = ""
469         res += colorcodes["blue"]
470         res += "*" * TEXTWIDTH
471         res += "\n"
472         res += title.center( \
473             TEXTWIDTH + \
474                 len(colorcodes["white"])) + \
475                     len(colorcodes["blue"]) + \
476                         "*")
477         res += "\n"
478         res += "*" * TEXTWIDTH
479         res += "\n"
480         res += colorcodes["white"]

```

```

        orderkeys = self.subtestorder.keys()
        orderkeys.sort()
        for key in orderkeys:
            label = self.subtestorder[key]
            subtest = self.subtests[label]
            res += str(subtest)

    487     res += "=" * TEXTWIDTH
    488     res += "\n"
    489     res += colorcodes["blue"]
    490     res += "*" * TEXTWIDTH
    491     res += "\n"
    492     res += colorcodes["white"]
    493     return res

    def getTeX(self):
        """Returns a string with a TeX table summarizing the tests."""
        title = self.title
        res = ""
        res += "\\begin{tabular}{|r|l|} \n"
        res += "\\hline \n"
        res += "\\multicolumn{2}{c}{%s} \\\\ \n" % title
        res += "\\hline \n"

        orderkeys = self.subtestorder.keys()
        orderkeys.sort()
        for key in orderkeys:
            label = self.subtestorder[key]
            subtest = self.subtests[label]
            res += subtest.getTeX()

        res += "\\hline \n"
    512     res += "\\end{tabular}"
    513     return res

    def save(self, filename = None, dir = None, quiet = False):
        """
        Saves the test object on a file.
        If filename is not given, a filename is created
        from the test title and the current time.
        """
        if not filename:
            filename = self.getTitle() + "." + str(time.time())
        if not dir:
            dir = "."

        completefilename = dir + os.sep + filename
    527
        try:
            outfile = open(completefilename, "w")
            pickle.dump(self, outfile)
            outfile.close()
        except:
            raise
        else:
            if not quiet:
                print "Test summary saved on file '%s'." % completefilename
    537     return completefilename

    def testload(filename, quiet = False):
        """
        Loads a (pickled) test summary from a file.
        """
        infile = open(filename, "r")
        test = pickle.load(infile)
        infile.close()
    547     if isinstance(test, Test):
            if not quiet:
                print "Test summary loaded from file '%s'." % filename
            return test
        else:
            raise InvalidCommandException, "Not a valid Test() instance"

```

```

557 if __name__ == "__main__":
561     # A new Test() instance
562     test = Test("Foo Bar")
563
564     # Initial setup
565     test.addSubtest('SetUp', task=True)
566     test.addSubtestTitle('SetUp', "Initial configuration")
567     test.begin('SetUp')
568     test.end('SetUp', TEST.OK)
569
570     # First sub-test
571     test.addSubtest('FirstTest') # labels may be strings or numbers
572     test.addSubtestTitle('FirstTest', "Just a test")
573     test.addSubtestDependency('FirstTest', 'SetUp') # 'FirstTest' depends on 'SetUp'
574
575     # begin the test
576     try:
577         test.begin('FirstTest')
578         # Collect some results during the sub-test
579         test.addResult('FirstTest', 'HelloInterval', 10)
580         test.addResult('FirstTest', 'DeadInterval', 40)
581         test.addResult('FirstTest', 'Flags', 'E+M')
582     except TestDependencyException:
583         # The dependencies were not met
584         test.end('FirstTest', TEST.SKIPPED)
585     except Exception, err:
586         # An error occurred
587         print type(err), err
588         test.end('FirstTest', TEST.FAILED)
589     except:
590         # Unexpected error
591         raise
592     else:
593         # The test succeeded
594         test.end('FirstTest', TEST.OK)
595
596     # Second sub-test
597     test.addSubtest(2)
598     test.addSubtestTitle(2, "Another test")
599     # begin the test
600     test.begin(2)
601     test.end(2, TEST.FAILED)
602     # end the test
603
604     # Third sub-test
605     test.addSubtestTitle('3', "And another one")
606     test.addSubtestDependency('3', 2)
607     # begin the test
608     try:
609         test.begin('3')
610         # Collect some results
611         test.addResult('3', 'Supercapsule', "OK!")
612     except TestDependencyException:
613         # The dependencies were not met
614         test.end('3', TEST.SKIPPED)
615     except Exception, err:
616         # An error occurred
617         print type(err), err
618         test.end('3', TEST.FAILED)
619     except:
620         # Unexpected error
621         raise
622     else:
623         # Test succeeded
624         test.end('3', TEST.OK)
625
626     # Output on the screen
627     print test
628     print test.getTeX()
629
630     # Output on a file
631     savedfile = test.save()

```

```

627     # Load from file
628     del test
629     test = testload(savedfile)
630     print test
631
632     test.save(dir = "/tmp")

```

A.5 The localconf module

Listing 8: The localconf module.

```

import pexpect

class LocalConfigurationException(Exception):
    "Error in the local machine configuration process."
    pass

def localcommand(command):
    "Execute a local configuration command."
    command_output, exitstatus = pexpect.run(command, withexitstatus=1)
    if exitstatus != 0:
        raise LocalConfigurationException(command)

```

B Source Code of the Tests

This appendix reports the integral source code of the tests described in section 3.

B.1 The Basic Test

The source code here reported is explained in section 3.1.

B.1.1 The Basic Test for Juniper J2320

Listing 9: The basic test script for Juniper J2320.

```

# Perform an ICMP connectivity test and verify the emission
# of correct OSPF Hello packets from a Juniper J2320 router

4  from scapy_ospf import *
from localconf import *
from juniperj2320 import *
from testsummary import *
import time

9   SERIALDEVICE = '/dev/ttyUSB0'
ROUTER_IP = '191.168.0.31'
ROUTER_MASK = '24'
ROUTER_USERNAME = 'root'
ROUTER_PASSWORD = 'secret'
ROUTER_INTERFACE = "ge-0/0/0"
LOCAL_INTERFACE = 'eth0'
LOCAL_IP = '191.168.0.32'
LOCAL_MASK = '24'
OSPF_AREA = '0.0.0.0'

TEST_OUTPUT_DIR = "./test-runs"

24  test = Test("Juniper J2320 Basic Test")
# The subtests
test.addSubtest('localconf', task = True)

```

```

    test.addSubtestTitle('localconf', "Local setup")
29  test.addSubtest('routerconf', task = True)
    test.addSubtestTitle('routerconf', "Router setup")
    # if local set-up was not successful do not configure the router
    test.addSubtestDependency('routerconf', 'localconf')

34  test.addSubtest('routerinfo', task = True)
    test.addSubtestTitle('routerinfo', "Retrieve router model information")
    test.addSubtestDependency('routerinfo', 'routerconf')

    test.addSubtest('icmp')
    test.addSubtestTitle('icmp', "ICMP connectivity test")
    test.addSubtestDependency('icmp', 'localconf')
    test.addSubtestDependency('icmp', 'routerconf')

    test.addSubtest('hello')
    test.addSubtestTitle('hello', "Emission of correct OSPF Hello packets")
    test.addSubtestDependency('hello', 'localconf')
    test.addSubtestDependency('hello', 'routerconf')

    test.addSubtest('ospfdisable', task = True)
    test.addSubtestTitle('ospfdisable', "Disable OSPF on the router")
    test.addSubtestDependency('ospfdisable', 'routerconf')

# Local machine configuration
try:
54    test.begin('localconf')
        localcommand("ip addr flush dev %s" % LOCAL_INTERFACE)
        localcommand("ip addr add %s/%s dev %s" % (LOCAL_IP, LOCAL_MASK,
                                                       LOCAL_INTERFACE))
        test.addResult('localconf', "Local IP Address", LOCAL_IP)
        test.addResult('localconf', "Local Netmask", "/" + LOCAL_MASK)
59        localcommand("ip link set %s up" % LOCAL_INTERFACE)
        test.addResult('localconf', "Local Interface", LOCAL_INTERFACE)

except:
    test.end('localconf', TEST_FAILED)
else:
64    test.end('localconf', TEST_OK)

# Now configure the router
try:
69    test.begin('routerconf')
        router = JuniperJ2320(SERIALDEVICE)

        # turn on logging
        logfile = open("%s/juniperj2320-%s.log" % (TEST_OUTPUT_DIR, time.time()),
                      "w")
74        router.logfile = logfile

        router.setUsername(ROUTER_USERNAME)
        router.setPassword(ROUTER_PASSWORD)

79        test.addResult('routerconf', "Serial Device", SERIALDEVICE)
        test.addResult('routerconf', "Router Username", ROUTER_USERNAME)
        test.addResult('routerconf', "Router Password", "***")

        router.gotoconf()
        router.sendcommand("delete interfaces %s unit 0 family inet" %
                           ROUTER_INTERFACE)
        router.confpromptexpect()
        router.sendcommand("set interfaces %s unit 0 family inet address %s/%s" \
                           % (ROUTER_INTERFACE, ROUTER_IP, ROUTER_MASK))

89        # OSPF configuration
        router.confpromptexpect()
        router.sendcommand("set routing-options router-id %s" % ROUTER_IP)
        router.confpromptexpect(timeout=10)
        router.sendcommand("set protocols ospf area %s interface %s enable" \
                           % (OSPF_AREA, ROUTER_INTERFACE))
        router.confpromptexpect(timeout=10)
        router.sendcommand("set protocols ospf enable")

```

```

99     # commit
100    router.commit()

101    test.addResult('routerconf', "Router Interface", ROUTER_INTERFACE)
102    test.addResult('routerconf', "Router IP Address", ROUTER_IP)
103    test.addResult('routerconf', "Router Netmask", "/" + ROUTER_MASK)
104    test.addResult('routerconf', "OSPF Area", OSPF_AREA)

105    except TestDependencyException:
106        # The dependencies were not met
107        test.end('routerconf', TEST_SKIPPED)
108    except Exception, err:
109        # An error occurred
110        print type(err), err
111        test.end('routerconf', TEST_FAILED)
112    except:
113        # Unexpected error
114        raise
115    else:
116        # The test succeeded
117        test.end('routerconf', TEST_OK)
118

119    # Retrieve router information
120    try:
121        test.begin('routerinfo')
122
123        router.gotocli()
124        router.clipromptexpect()
125        router.sendcommand("show version")
126        router.readuntil('\n')
127
128        routerhostname = router.readuntil('\n')
129        print routerhostname
130
131        routermode = router.readuntil('\n')
132        print routermode
133
134        routeros = router.readuntil('\n')
135        print routeros
136
137        test.addResult('routerinfo', "Router Hostname", routerhostname)
138        test.addResult('routerinfo', "Router Model", routermode)
139        test.addResult('routerinfo', "Router OS", routeros)

140    except TestDependencyException:
141        # The dependencies were not met
142        test.end('routerinfo', TEST_SKIPPED)
143    except Exception, err:
144        # An error occurred
145        print type(err), err
146        test.end('routerinfo', TEST_FAILED)
147    except:
148        # Unexpected error
149        raise
150    else:
151        # The test succeeded
152        test.end('routerinfo', TEST_OK)

153    # now check connectivity using icmp
154    try:
155        test.begin('icmp')
156        test.announce("Checking connectivity using ICMP")
157
158        conf.iface = LOCAL_INTERFACE
159
160        # an icmp echo-request packet
161        icmp_echo_request = IP(dst=ROUTER_IP)/ICMP()/"XXXXXXXXXXXXXXXXXX"
162
163        print "Sending an ICMP echo-request packet"
164
165        assert(icmp_echo_request != None)

```

```

    icmp_echo_request.show()

174    # send the packet and get the reply
    icmp_echo_reply = sr1(icmp_echo_request, timeout = 10)

    assert(icmp_echo_reply != None)
    print "ICMP echo-reply received"
    icmp_echo_reply.show()

    assert(icmp_echo_reply.type == 0)

184    except TestDependencyException:
        # The dependencies were not met
        test.end('icmp', TEST_SKIPPED)
    except Exception, err:
        print type(err), err
        test.end('icmp', TEST_FAILED)
189    except:
        # Unexpected error
        raise
    else:
        # The test succeeded
        test.end('icmp', TEST_OK)

# Now sniff an ospf hello packet
try:
    test.begin('hello')
    test.announce("Trying to sniff an OSPF Hello Packet...")

    snifffedpackets = sniff(count=1, lfilter = lambda x: x.haslayer(OSPF_Hello),
                             timeout=60)
    assert(len(snifffedpackets) > 0)
    snifffedpackets.show()
    p = snifffedpackets[0]
    pospf = p.getlayer(OSPF_Hdr)
    pospf.display()

    test.addResult('hello', 'OSPF Type', pospf.type)
    test.addResult('hello', 'OSPF Version', pospf.version)
    test.addResult('hello', 'OSPF Source address', pospf.src)
    test.addResult('hello', 'OSPF Area', pospf.area)
    test.addResult('hello', 'OSPF Auth Type', pospf.authtype)
    test.addResult('hello', 'OSPF Hello Interval', pospf.hellointerval)
    test.addResult('hello', 'OSPF Hello Dead Interval', pospf.deadinterval)
    test.addResult('hello', 'OSPF Hello Options', pospf.options)
    test.addResult('hello', 'OSPF Hello NetMask', pospf.mask)
    test.addResult('hello', 'OSPF Hello Designated Router', pospf.router)
    test.addResult('hello', 'OSPF Hello Backup Router', pospf.backup)
    test.addResult('hello', 'OSPF Hello Neighbors', pospf.neighbor)
    assert(pospf.type == 1)
    assert(pospf.version == 2)
    assert(pospf.src == ROUTER_IP)
    assert(pospf.area == OSPF_AREA)

224    except TestDependencyException:
        # The dependencies were not met
        test.end('hello', TEST_SKIPPED)
    except Exception, err:
        # An error occurred
        print type(err), err
        test.end('hello', TEST_FAILED)
    except:
        # Unexpected error
        raise
    else:
        # The test succeeded
        test.end('hello', TEST_OK)

239    # now disable ospf on the router
try:
    test.begin('ospfdisable')
    router.gotoconf()

```

```

249     router.confpromptexpect()
250     router.sendcommand("set protocols ospf disable")
251     router.commit()
252     router.gotologinscreen()
253
254     except TestDependencyException:
255         # The dependencies were not met
256         test.end('ospfdisable', TEST_SKIPPED)
257     except Exception, err:
258         # An error occurred
259         print type(err), err
260         test.end('ospfdisable', TEST_FAILED)
261     except:
262         # Unexpected error
263         raise
264     else:
265         # The test succeeded
266         test.end('ospfdisable', TEST_OK)
267
268     # turn off logging
269     logfile.close()
270
271     print test
272     test.save(dir = TEST_OUTPUT_DIR)

```

B.1.2 The Basic Test for Cisco 2811

Listing 10: The basic test script for Cisco 2811.

```

1  # Perform an ICMP connectivity test and verify the emission
# of correct OSPF Hello packets from a Juniper J2320 router
2
3  from scapy_ospf import *
4  from localconf import *
5  from cisco2811 import *
6  from testsummary import *
7  import time
8
9  SERIALDEVICE = '/dev/ttyUSB0'
10 ROUTER_IP = '191.168.0.31'
11 ROUTER_MASK = '255.255.255.0'
12 ROUTER_USERNAME = 'admin'
13 ROUTER_PASSWORD = 'secret'
14 ROUTER_HOSTNAME = 'cisco2'
15 ROUTER_INTERFACE = "FastEthernet 0/0"
16 LOCAL_INTERFACE = 'eth0'
17 LOCAL_IP = '191.168.0.32'
18 LOCAL_MASK = '24'
19 OSPF_AREA = '0.0.0.0'
20
21 TEST_OUTPUT_DIR = "./test-runs"
22
23 test = Test("Cisco 2811 Basic Test")
24
25 # The subtests
26 test.addSubtest('localconf', task = True)
27 test.addSubtestTitle('localconf', "Local setup")
28
29 test.addSubtest('routerconf', task = True)
30 test.addSubtestTitle('routerconf', "Router setup")
31 # if local set-up was not successful do not configure the router
32 test.addSubtestDependency('routerconf', 'localconf')
33
34 test.addSubtest('routerinfo', task = True)
35 test.addSubtestTitle('routerinfo', "Retrieve router model information")
36 test.addSubtestDependency('routerinfo', 'routerconf')
37
38 test.addSubtest('icmp')
39 test.addSubtestTitle('icmp', "ICMP connectivity test")
40 test.addSubtestDependency('icmp', 'localconf')
41 test.addSubtestDependency('icmp', 'routerconf')

```

```

46     test.addSubtest('hello')
47     test.addSubtestTitle('hello', "Emission of correct OSPF Hello packets")
48     test.addSubtestDependency('hello', 'localconf')
49     test.addSubtestDependency('hello', 'routerconf')

50     test.addSubtest('ospfdisable', task = True)
51     test.addSubtestTitle('ospfdisable', "Disable OSPF on the router")
52     test.addSubtestDependency('ospfdisable', 'routerconf')

# Local machine configuration
53     try:
54         test.begin('localconf')
55         localcommand("ip addr flush dev %s" % LOCAL_INTERFACE)
56         localcommand("ip addr add %s/%s dev %s" % (LOCAL_IP, LOCAL_MASK,
57             LOCAL_INTERFACE))
58         test.addResult('localconf', "Local IP Address", LOCAL_IP)
59         test.addResult('localconf', "Local Netmask", "/" + LOCAL_MASK)
60         localcommand("ip link set %s up" % LOCAL_INTERFACE)
61         test.addResult('localconf', "Local Interface", LOCAL_INTERFACE)

62     except:
63         test.end('localconf', TEST_FAILED)
64     else:
65         test.end('localconf', TEST_OK)

# Now configure the router
66     try:
67         test.begin('routerconf')
68         router = Cisco2811(SERIALDEVICE)

69         # turn on logging
70         logfile = open("%s/cisco2811-%s.log" % (TEST_OUTPUT_DIR, time.time()), "w"
71         )
72         router.logfile = logfile

73         router.setUsername(ROUTER_USERNAME)
74         router.setPassword(ROUTER_PASSWORD)
75         router.setHostname(ROUTER_HOSTNAME)

76         test.addResult('routerconf', "Serial Device", SERIALDEVICE)

77         router.gotoconfig()
78         test.addResult('routerconf', "Router Username", ROUTER_USERNAME)
79         test.addResult('routerconf', "Router Password", "***")

80         router.gotoconfig()
81         router.sendcommand("interface %s" % ROUTER_INTERFACE)
82         router.config.promptexpect("if")
83         router.sendcommand("ip address %s %s" % (ROUTER_IP, ROUTER_MASK))
84         router.config.promptexpect("if")
85         router.sendcommand("no shutdown")
86         router.config.promptexpect("if")
87         router.sendcommand("end")
88         router.enabledpromptexpect()

89     except:
90         test.end('routerconf', TEST_FAILED)
91         test.addResult('routerconf', "Router Interface", ROUTER_INTERFACE)
92         test.addResult('routerconf', "Router IP Address", ROUTER_IP)
93         test.addResult('routerconf', "Router Netmask", "/" + ROUTER_MASK)
94         test.addResult('routerconf', "OSPF Area", OSPF_AREA)

95     # OSPF configuration
96     router.gotoconfig()
97     router.sendcommand("router ospf 100")
98     router.config.promptexpect("router")
99     router.sendcommand("network %s 255.255.255.255 area %s" %(ROUTER_IP,
100         OSPF_AREA))
101     router.config.promptexpect("router")
102     router.sendcommand("end")

103     # write configuration
104     router.write()

105     test.addResult('routerconf', "Router Interface", ROUTER_INTERFACE)
106     test.addResult('routerconf', "Router IP Address", ROUTER_IP)
107     test.addResult('routerconf', "Router Netmask", "/" + ROUTER_MASK)
108     test.addResult('routerconf', "OSPF Area", OSPF_AREA)

109 except TestDependencyException:

```

```

# The dependencies were not met
test.end('routerconf', TEST_SKIPPED)
116 except Exception, err:
# An error occurred
print type(err), err
test.end('routerconf', TEST_FAILED)
except:
# Unexpected error
raise
else:
# The test succeeded
test.end('routerconf', TEST_OK)
126

# Retrieve router information
try:
    test.begin('routerinfo')
131
    router.gotocli()
    router.clipromptexpect()
    router.sendcommand("show version")
    router.readuntil('\n')
136
    routerinfo = router.readuntil(router.cliprompt)
    print routerinfo

    routerinfo = "\n" + routerinfo
141
    test.addResult('routerinfo', "Router Information", routerinfo)

except TestDependencyException:
# The dependencies were not met
test.end('routerinfo', TEST_SKIPPED)
except Exception, err:
# An error occurred
print type(err), err
test.end('routerinfo', TEST_FAILED)
except:
# Unexpected error
raise
else:
# The test succeeded
test.end('routerinfo', TEST_OK)
156

# now check connectivity using icmp
try:
    test.begin('icmp')
    test.announce("Checking connectivity using ICMP")
    conf.iface = LOCAL_INTERFACE
161
    # an icmp echo-request packet
    icmp_echo_request = IP(dst=ROUTER.IP)/ICMP()/"XXXXXXXXXXXXXXXXXX"

    print "Sending an ICMP echo-request packet"
166
    assert(icmp_echo_request != None)

    icmp_echo_request.show()

    # send the packet and get the reply
    171 icmp_echo_reply = sr1(icmp_echo_request, timeout = 10)

    assert(icmp_echo_reply != None)
    print "ICMP echo-reply received"
    icmp_echo_reply.show()
176
    assert(icmp_echo_reply.type == 0)

except TestDependencyException:
# The dependencies were not met
test.end('icmp', TEST_SKIPPED)
except Exception, err:

```

```

    print type(err), err
    test.end('icmp', TEST_FAILED)
except:
    # Unexpected error
    raise
else:
    # The test succeeded
    test.end('icmp', TEST_OK)

# Now sniff an ospf hello packet
try:
    test.begin('hello')
    test.announce("Trying to sniff an OSPF Hello Packet...")

    sniffeedpackets = sniff(count=1, lfilter = lambda x: x.haslayer(OSPF_Hello),
                             timeout=60)
    assert(len(sniffeedpackets) > 0)
    sniffeedpackets.show()
    p = sniffeedpackets[0]
    pospf = p.getlayer(OSPF_Hdr)
    pospf.display()

    test.addResult('hello', 'OSPF Type', pospf.type)
    test.addResult('hello', 'OSPF Version', pospf.version)
    test.addResult('hello', 'OSPF Source address', pospf.src)
    test.addResult('hello', 'OSPF Area', pospf.area)
    test.addResult('hello', 'OSPF Auth Type', pospf.authtype)
    test.addResult('hello', 'OSPF Hello Interval', pospf.hellointerval)
    test.addResult('hello', 'OSPF Hello Dead Interval', pospf.deadinterval)
    test.addResult('hello', 'OSPF Hello Options', pospf.options)
    test.addResult('hello', 'OSPF Hello NetMask', pospf.mask)
    test.addResult('hello', 'OSPF Hello Designated Router', pospf.router)
    test.addResult('hello', 'OSPF Hello Backup Router', pospf.backup)
    test.addResult('hello', 'OSPF Hello Neighbors', pospf.neighbor)
    assert(pospf.type == 1)
    assert(pospf.version == 2)
    assert(pospf.src == ROUTER_IP)
    assert(pospf.area == OSPF_AREA)
except TestDependencyException:
    # The dependencies were not met
    test.end('hello', TEST_SKIPPED)
except Exception, err:
    # An error occurred
    print type(err), err
    test.end('hello', TEST_FAILED)
except:
    # Unexpected error
    raise
else:
    # The test succeeded
    test.end('hello', TEST_OK)

# now disable ospf on the router
try:
    test.begin('ospfdisable')
    router.gotoconfig()
    router.sendcommand("no router ospf 100")
    router.configpromptexpect()
    # Write configuration
    router.write()
    router.gotologinscreen()
except TestDependencyException:
    # The dependencies were not met
    test.end('ospfdisable', TEST_SKIPPED)
except Exception, err:
    # An error occurred
    print type(err), err
    test.end('ospfdisable', TEST_FAILED)
except:
    # Unexpected error

```

```

261     raise
else:
    # The test succeeded
    test.end('ospfdisable', TEST.OK)

266 # turn off logging
logfile.close()

print test
test.save(dir = TEST.OUTPUT.DIR)

```

B.2 The Adjacency Initial Forming Test

For an explanation of the following source code please refer to section 3.2.

B.2.1 The Adjacency Initial Forming Test for Juniper J2320

Listing 11: The adjacency initial forming test script for Juniper J2320.

```

from scapy_ospf import *
from localconf import *
from juniperj2320 import *
from testsummary import *
import sys
import traceback

# Verify that the router behaves as in section 10.10 of RFC 2328 (OSPFv2),
# where an adjacency forming example is shown

AllSPFRouters = '224.0.0.5'

SERIALDEVICE = '/dev/ttyUSB0'
ROUTER_IP = '191.168.0.31'
ROUTER_MASK = '24'
ROUTER_USERNAME = 'root'
ROUTER_PASSWORD = 'secret'
ROUTER_INTERFACE = "ge-0/0/0"
LOCAL_INTERFACE = 'eth0'
LOCAL_IP = '191.168.0.32'
LOCAL_MASK = '24'
LOCAL_FULL_MASK = '255.255.255.0'
OSPF_AREA = '0.0.0.0'
ROUTER_PRIORITY = 100
LOCAL_PRIORITY = 200

TEST_RUN_DIR = "./test-runs"

29 test = Test("Juniper J2320 RFC2328 Section 10.10 Example Conformance")

test.addSubtest('localconf', task = True)
test.addSubtestTitle('localconf', "Local setup")

34 test.addSubtest('routerconf', task = True)
test.addSubtestTitle('routerconf', "Router setup")
# if local set-up was not successful do not configure the router
test.addSubtestDependency('routerconf', 'localconf')

39 test.addSubtest('10.10')
test.addSubtestTitle('10.10', "Begin the formation of an adjacency")
test.addSubtestDependency('10.10', 'localconf')
test.addSubtestDependency('10.10', 'routerconf')

44 test.addSubtest('ospfdisable', task = True)
test.addSubtestTitle('ospfdisable', "Disable OSPF on the router")
test.addSubtestDependency('ospfdisable', 'routerconf')

49 test.addSubtest('finallocalconf', task = True)
test.addSubtestTitle('finallocalconf', "Restore local configuration")
test.addSubtestDependency('finallocalconf', 'localconf')

```

```

# Local machine configuration
try:
    test.begin('localconf')
    localcommand("ip addr flush dev %s" % LOCAL_INTERFACE)
    localcommand("ip addr add %s/%s dev %s" % (LOCAL_IP, LOCAL_MASK,
                                                LOCAL_INTERFACE))
    test.addResult('localconf', "Local IP Address", LOCAL_IP)
    test.addResult('localconf', "Local Netmask", "/" + LOCAL_MASK)
    localcommand("ip link set %s up" % LOCAL_INTERFACE)
    test.addResult('localconf', "Local Interface", LOCAL_INTERFACE)

    localcommand("ip route add 224.0.0.0/8 dev %s" % LOCAL_INTERFACE)
    # avoid protocol-unreachable messages from this host
    localcommand("iptables -A OUTPUT -p icmp -m icmp --icmp-type protocol-
unreachable -j DROP")

    # scapy interface
    conf iface = LOCAL_INTERFACE

    # resync scapy with the local routing table
    conf.route.resync()

except:
    test.end('localconf', TEST_FAILED)
else:
    test.end('localconf', TEST_OK)

# Now configure the router
try:
    test.begin('routerconf')
    router = JuniperJ2320(SERIALDEVICE)
    router.setUsername(ROUTER.USERNAME)
    router.setPassword(ROUTER.PASSWORD)

    test.addResult('routerconf', "Serial Device", SERIALDEVICE)
    test.addResult('routerconf', "Router Username", ROUTER.USERNAME)
    test.addResult('routerconf', "Router Password", "***")

    router.gotoconf()
    router.sendcommand("delete interfaces %s unit 0 family inet" %
                      ROUTER_INTERFACE)
    router.confpromptexpect()
    router.sendcommand("set interfaces %s unit 0 family inet address %s/%s" \
                      % (ROUTER_INTERFACE, ROUTER_IP, ROUTER_MASK))
    router.sendcommand("set interfaces %s enable" % (ROUTER_INTERFACE))

# OSPF configuration
    router.confpromptexpect()
    router.sendcommand("set routing-options router-id %s" % ROUTER_IP)
    router.confpromptexpect(timeout=10)
    router.sendcommand("set protocols ospf area %s interface %s priority %s" \
                      % (OSPF_AREAS, ROUTER_INTERFACE, ROUTER_PRIORITY))
    router.confpromptexpect(timeout=10)
    router.sendcommand("set protocols ospf area %s interface %s enable" \
                      % (OSPF_AREAS, ROUTER_INTERFACE))
    router.confpromptexpect(timeout=10)
    router.sendcommand("set protocols ospf enable")

    # commit
    router.commit()

    test.addResult('routerconf', "Router Interface", ROUTER_INTERFACE)
    test.addResult('routerconf', "Router IP Address", ROUTER_IP)
    test.addResult('routerconf', "Router Netmask", "/" + ROUTER_MASK)
    test.addResult('routerconf', "Router OSPF Priority", ROUTER_PRIORITY)
    test.addResult('routerconf', "OSPF Area", OSPF_AREAS)

except TestDependencyException:
    # The dependencies were not met
    test.end('routerconf', TEST_SKIPPED)
except Exception, err:
    # An error occurred
    print type(err), err

```

```

    traceback.print_tb(sys.exc_info()[2])
    test.end('routerconf', TEST_FAILED)
124 except:
# Unexpected error
    raise
else:
# The test succeeded
129     test.end('routerconf', TEST_OK)

# Now begin the formation of an adjacency
#
134 #          +---+
#          |ROU|
#          +---+
#          |
#          Down
#          +---+           +---+
#          |LOC|           +---+
#          +---+
#          Hello (DR=0,seen=0)           Down
#          +---+           +---+
#          <Hello (DR=LOC,seen=ROU,...)>
#          +---+           +---+
#          ExStart           Init
#          +---+           +---+
#          D-D (Seq=x,I,M,Master)           >
#          +---+           +---+
144 #
# try:
#     test.begin('10.10')

149     test.announce("Wait for an OSPF Hello from the router")
# Wait for an OSPF Hello from the router
#           Hello (DR=0,seen=0)
#           +---+
# sniffedpackets = sniff(count=1, lfilter = lambda x: x.haslayer(OSPF_Hello)
#           , timeout=60)
154     assert len(sniffedpackets) > 0
sniffedpackets.show()
rp1 = sniffedpackets[0]
pospf = rp1.getlayer(OSPF_Hdr)
pospf.display()
159     assert pospf.type == 1
assert pospf.version == 2
assert pospf.src == ROUTER_IP
assert pospf.area == OSPF_AREA
assert pospf.prio == ROUTER_PRIORITY
164     assert pospf.authtype == 0

test.announce("Reply to the Hello including the router as neighbor")
# Reply to the Hello including the router as neighbor
#           Hello (DR=LOC,seen=ROU,...)
#           +---+
#           <
# p1 = IP()/OSPF_Hdr()/OSPF_Hello()

169     p1[IP].src = LOCAL_IP
p1[IP].dst = AllSPFRouters
174     p1[OSPF_Hdr].src = LOCAL_IP
p1[OSPF_Hdr].len = 48 # scapy_ospf bug

179     p1[OSPF_Hello].mask = LOCAL_FULL_MASK
p1[OSPF_Hello].options = 'E'
p1[OSPF_Hello].hellointerval = pospf.hellointerval
p1[OSPF_Hello].deadinterval = pospf.deadinterval
p1[OSPF_Hello].prio = LOCAL_PRIORITY
p1[OSPF_Hello].router = LOCAL_IP # DR
p1[OSPF_Hello].neighbor = ROUTER_IP # seen
184     send(p1)

test.announce("Wait for a Database Description Packet")
# Wait for a Database Description
#           D-D (Seq=x,I,M,Master)
#           +---+
# sniffedpackets = sniff(count = 1, lfilter = lambda x: x.haslayer(
#           OSPF_DBDesc), timeout = 30)
189     assert len(sniffedpackets) > 0

```

```

194     sniffedpackets.show()
195     rp2 = sniffedpackets[0]
196     pospf = rp2.getlayer(OSPF_Hdr)
197     pospf.display()
198     assert pospf.type == 2
199     assert pospf.version == 2
200     assert pospf.src == ROUTER_IP
201     assert pospf.area == OSPF_AREA
202     assert pospf.authtype == 0
203     assert pospf.dbdescr == 7
204
205     test.announce("Correct Database Description Packet received")
206
207     except TestDependencyException:
208         # The dependencies were not met
209         test.end('10.10', TEST_SKIPPED)
210     except Exception, err:
211         # An error occurred
212         print type(err), err
213         traceback.print_tb(sys.exc_info() [2])
214         test.end('10.10', TEST_FAILED)
215     except:
216         # Unexpected error
217         raise
218     else:
219         # The test succeeded
220         test.end('10.10', TEST_OK)
221
222     # now disable ospf on the router
223     try:
224         test.begin('ospfdisable')
225         router.confpromptexpect()
226         router.sendcommand("set protocols ospf disable")
227         router.commit()
228         router.gotologinscreen()
229
230     except TestDependencyException:
231         # The dependencies were not met
232         test.end('ospfdisable', TEST_SKIPPED)
233     except Exception, err:
234         # An error occurred
235         print type(err), err
236         traceback.print_tb(sys.exc_info() [2])
237         test.end('ospfdisable', TEST_FAILED)
238     except:
239         # Unexpected error
240         raise
241     else:
242         # The test succeeded
243         test.end('ospfdisable', TEST_OK)
244
245
246     try:
247         test.begin('finallocalconf')
248
249         localcommand("ip route del 224.0.0.0/8 dev %s" % LOCAL_INTERFACE)
250         localcommand("iptables -D OUTPUT -p icmp -m icmp --icmp-type protocol-
251             unreachable -j DROP")
252
253     except TestDependencyException:
254         # The dependencies were not met
255         test.end('finallocalconf', TEST_SKIPPED)
256     except Exception, err:
257         # An error occurred
258         print type(err), err
259         traceback.print_tb(sys.exc_info() [2])
260         test.end('finallocalconf', TEST_FAILED)
261     except:
262         # Unexpected error
263         raise
264     else:

```

```

# The test succeeded
test.end('finallocalconf', TEST_OK)
269
print test
test.save(dir = TEST_RUN_DIR)

```

B.2.2 The Adjacency Initial Forming Test for Cisco 2811

Listing 12: The adjacency initial forming test script for Cisco 2811.

```

from scapy_ospf import *
from localconf import *
from cisco2811 import *
from testsummary import *
import sys
import traceback

8 # Verify that the router behaves as in section 10.10 of RFC 2328 (OSPFv2),
# where an adjacency forming example is shown

AllSPFRouters = '224.0.0.5'

13 SERIALDEVICE = '/dev/ttyUSB0'
ROUTER_IP = '191.168.0.31'
ROUTER_MASK = '255.255.255.0'
ROUTER_USERNAME = 'admin'
ROUTER_PASSWORD = 'secret',
18 ROUTER_HOSTNAME = 'cisco2'
ROUTER_INTERFACE = "FastEthernet 0/0"
LOCAL_INTERFACE = 'eth0'
LOCAL_IP = '191.168.0.32',
LOCAL_MASK = '24',
23 LOCAL_FULL_MASK = '255.255.255.0'
OSPF_AREA = '0.0.0.0'
ROUTER_PRIORITY = 100
LOCAL_PRIORITY = 200

28 TEST_RUN_DIR = "./test-runs"

test = Test("Cisco 2811 RFC2328 Section 10.10 Example Conformance")

test.addSubtest('localconf', task = True)
33 test.addSubtestTitle('localconf', "Local setup")

test.addSubtest('routerconf', task = True)
test.addSubtestTitle('routerconf', "Router setup")
# if local set-up was not successful do not configure the router
38 test.addSubtestDependency('routerconf', 'localconf')

test.addSubtest('10.10')
test.addSubtestTitle('10.10', "Begin the formation of an adjacency")
test.addSubtestDependency('10.10', 'localconf')
43 test.addSubtestDependency('10.10', 'routerconf')

test.addSubtest('ospfdisable', task = True)
test.addSubtestTitle('ospfdisable', "Disable OSPF on the router")
test.addSubtestDependency('ospfdisable', 'routerconf')
48

test.addSubtest('finallocalconf', task = True)
test.addSubtestTitle('finallocalconf', "Restore local configuration")
test.addSubtestDependency('finallocalconf', 'localconf')

53 # Local machine configuration
try:
    test.begin('localconf')
    localcommand("ip addr flush dev %s" % LOCAL_INTERFACE)
    localcommand("ip addr add %s/%s dev %s" % (LOCAL_IP, LOCAL_MASK,
        LOCAL_INTERFACE))
    test.addResult('localconf', "Local IP Address", LOCAL_IP)
    test.addResult('localconf', "Local Netmask", "/" + LOCAL_MASK)
    localcommand("ip link set %s up" % LOCAL_INTERFACE)
58

```

```

    test.addResult('localconf', "Local Interface", LOCAL_INTERFACE)

63   localcommand("ip route add 224.0.0.0/8 dev %s" % LOCAL_INTERFACE)
# avoid protocol-unreachable messages from this host
64   localcommand("iptables -A OUTPUT -p icmp --icmp-type protocol-
unreachable -j DROP")

68   # scapy interface
69   conf.iface = LOCAL_INTERFACE

70   # resync scapy with the local routing table
71   conf.route.resync()

73   except:
74       test.end('localconf', TEST_FAILED)
75   else:
76       test.end('localconf', TEST_OK)

78   # Now configure the router
79   try:
80       test.begin('routerconf')
81       router = Cisco2811(SERIALDEVICE)
82       router.setUsername(ROUTER_USERNAME)
83       router.setPassword(ROUTER_PASSWORD)
84       router.setHostname(ROUTER_HOSTNAME)

85       test.addResult('routerconf', "Serial Device", SERIALDEVICE)
86       test.addResult('routerconf', "Router Username", ROUTER_USERNAME)
87       test.addResult('routerconf', "Router Password", "***")
88       test.addResult('routerconf', "Router Hostname", ROUTER_HOSTNAME)

89       router.gotoconfig()
90       router.sendcommand("interface %s" % ROUTER_INTERFACE)
91       router.config.promptexpect("if")
92       router.sendcommand("ip address %s %s" % (ROUTER_IP, ROUTER_MASK))
93       router.config.promptexpect("if")
94       router.sendcommand("no shutdown")
95       router.config.promptexpect("if")
96       router.sendcommand("end")
97       router.enabledpromptexpect()

98       # OSPF configuration
99       router.gotoconfig()
100      router.sendcommand("router ospf 100")
101      router.config.promptexpect("router")
102      router.sendcommand("network %s 255.255.255.255 area %s" %(ROUTER_IP,
103          OSPF_AREA))
104      router.config.promptexpect("router")
105      router.sendcommand("end")
106      router.enabledpromptexpect()

107      router.gotoconfig()
108      router.sendcommand("interface %s" % ROUTER_INTERFACE)
109      router.config.promptexpect("if")
110      router.sendcommand("ip ospf priority %s" % ROUTER_PRIORITY)
111      router.config.promptexpect("if")
112      router.sendcommand("end")
113      router.enabledpromptexpect()

114      test.addResult('routerconf', "Router Interface", ROUTER_INTERFACE)
115      test.addResult('routerconf', "Router IP Address", ROUTER_IP)
116      test.addResult('routerconf', "Router Netmask", "/" + ROUTER_MASK)
117      test.addResult('routerconf', "Router OSPF Priority", ROUTER_PRIORITY)
118      test.addResult('routerconf', "OSPF Area", OSPF_AREA)

119      except TestDependencyException:
120          # The dependencies were not met
121          test.end('routerconf', TEST_SKIPPED)
122      except Exception, err:
123          # An error occurred
124          print type(err), err
125          traceback.print_tb(sys.exc_info()[2])
126          test.end('routerconf', TEST_FAILED)

```

```

133     except:
134         # Unexpected error
135         raise
136     else:
137         # The test succeeded
138         test.end('routerconf', TEST_OK)

139
140     # Now begin the formation of an adjacency
141     #
142     #          +---+
143     #          |ROU|
144     #          +---+
145     #
146     #          Down
147     #          Hello (DR=0,seen=0)
148     #          +---+----->
149     #          Hello (DR=LOC,seen=ROU,...)
150     #          +---+-----<
151     #          ExStart
152     #          D-D (Seq=x,l,M,Master)
153     #          +---+----->
154
155     try:
156         test.begin('10.10')

157         test.announce("Wait for an OSPF Hello from the router")
158         # Wait for an OSPF Hello from the router
159         #           Hello (DR=0,seen=0)
160         #
161         sniffedpackets = sniff(count=1, lfilter = lambda x: x.haslayer(OSPF_Hello),
162                               timeout=60)
163         assert len(sniffedpackets) > 0
164         sniffedpackets.show()
165         rp1 = sniffedpackets[0]
166         pospf = rp1.getlayer(OSPF_Hdr)
167         pospf.display()
168         assert pospf.type == 1
169         assert pospf.version == 2
170         assert pospf.src == ROUTER_IP
171         assert pospf.area == OSPF_AREA
172         assert pospf.prio == ROUTER_PRIORITY
173         assert pospf.authtype == 0

174         test.announce("Reply to the Hello including the router as neighbor")
175         # Reply to the Hello including the router as neighbor
176         #           Hello (DR=LOC,seen=ROU,...)
177         #
178         p1 = IP() / OSPF_Hdr() / OSPF_Hello()
179
180         p1[IP].src = LOCAL_IP
181         p1[IP].dst = AllSPFRouters
182
183         p1[OSPF_Hdr].src = LOCAL_IP
184         p1[OSPF_Hdr].len = 48 # scapy_ospf bug

185         p1[OSPF_Hello].mask = LOCAL_FULL_MASK
186         p1[OSPF_Hello].options = 'E'
187         p1[OSPF_Hello].hellointerval = pospf.hellointerval
188         p1[OSPF_Hello].deadinterval = pospf.deadinterval
189         p1[OSPF_Hello].prio = LOCAL_PRIORITY
190         p1[OSPF_Hello].router = LOCAL_IP # DR
191         p1[OSPF_Hello].neighbor = ROUTER_IP # seen

192         send(p1)

193         test.announce("Wait for a Database Description Packet")
194         # Wait for a Database Description
195         #           D-D (Seq=x,l,M,Master)
196         #
197         sniffedpackets = sniff(count = 1, lfilter = lambda x: x.haslayer(
198             OSPF_DBDesc), timeout = 30)
199         assert len(sniffedpackets) > 0
200         sniffedpackets.show()
201         rp2 = sniffedpackets[0]

```

```

    pospf = rp2.getlayer(OSPF_Hdr)
    pospf.display()
    assert pospf.type == 2
    assert pospf.version == 2
    assert pospf.src == ROUTER_IP
    assert pospf.area == OSPF_AREA
    assert pospf.authtype == 0
    assert pospf.dbdescr == 7
213
    test.announce("Correct Database Description Packet received")

except TestDependencyException:
    # The dependencies were not met
    test.end('10.10', TEST_SKIPPED)
except Exception, err:
    # An error occurred
    print type(err), err
    traceback.print_tb(sys.exc_info() [2])
    test.end('10.10', TEST_FAILED)
except:
    # Unexpected error
    raise
else:
    # The test succeeded
    test.end('10.10', TEST_OK)

# now disable ospf on the router
233 try:
    test.begin('ospfdisable')
    router.gotoconfig()
    router.sendcommand("no router ospf 100")
    router.configpromptexpect()
    router.gotologinscreen()

except TestDependencyException:
    # The dependencies were not met
    test.end('ospfdisable', TEST_SKIPPED)
243 except Exception, err:
    # An error occurred
    print type(err), err
    traceback.print_tb(sys.exc_info() [2])
    test.end('ospfdisable', TEST_FAILED)
except:
    # Unexpected error
    raise
else:
    # The test succeeded
253 test.end('ospfdisable', TEST_OK)

258 try:
    test.begin('finallocalconf')

    localcommand("ip route del 224.0.0.0/8 dev %s" % LOCAL_INTERFACE)
    localcommand("iptables -D OUTPUT -p icmp -m icmp --icmp-type protocol-
                 unreachable -j DROP")

except TestDependencyException:
    # The dependencies were not met
    test.end('finallocalconf', TEST_SKIPPED)
except Exception, err:
    # An error occurred
    print type(err), err
    traceback.print_tb(sys.exc_info() [2])
    test.end('finallocalconf', TEST_FAILED)
except:
    # Unexpected error
    raise
else:
    # The test succeeded
    test.end('finallocalconf', TEST_OK)

```

```
print test
test.save(dir = TEST_RUN_DIR)
```

Listings

1	A simple script using the scapy module	3
2	A simple script using the pexpect module	4
3	A simple script using the pySerial module	4
4	The serialrouter module.	32
5	The juniperj2320 module.	33
6	The cisco2811 module.	35
7	The testsummary module.	38
8	The localconf module.	47
9	The basic test script for Juniper J2320.	47
10	The basic test script for Cisco 2811.	51
11	The adjacency initial forming test script for Juniper J2320.	55
12	The adjacency initial forming test script for Cisco 2811.	59

List of Tables

1	The summary of the Basic Test performed on a Juniper J2320 router.	17
2	The summary of the Basic Test performed on a Cisco 2811 router.	21
3	The summary of the Adjacency Initial Forming Test performed on a Juniper J2320 router.	29
4	The summary of the Adjacency Initial Forming Test performed on a Cisco 2811 router.	29

List of Figures

1	Class derivation diagram.	5
2	Connections between computer and router equipment for test execution.	11

Copyright

The source code reported in listings 4, 5, 6, 7, 9, 10, 11 and 12 is Copyright © Claudio Pisa 2008 and released under the GNU Public License version 3 or subsequent [12].