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

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

```
from scapy import *

# Forge an ICMP echo-request to the destination 192.168.1.1
p = IP(dst='192.168.1.1')/ICMP()
p[ICMP].type = 8 # echo-request

# Send the request and receive the reply
q = sr1(p)

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.
import pexpect
child = pexpect.spawn ('ftp ftp.openbsd.org')
child.expect ('Name .*: ')
child.sendline ('anonymous')
child.expect ('Password:')
child.sendline ('noah@example.com')
child.expect ('ftp> ')
child.sendline ('cd pub')
child.expect('ftp> ')
child.sendline ('get ls-lR.gz')
child.expect('ftp> ')
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)

s.write("ATZ\n")

readok = s.read(2)

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.

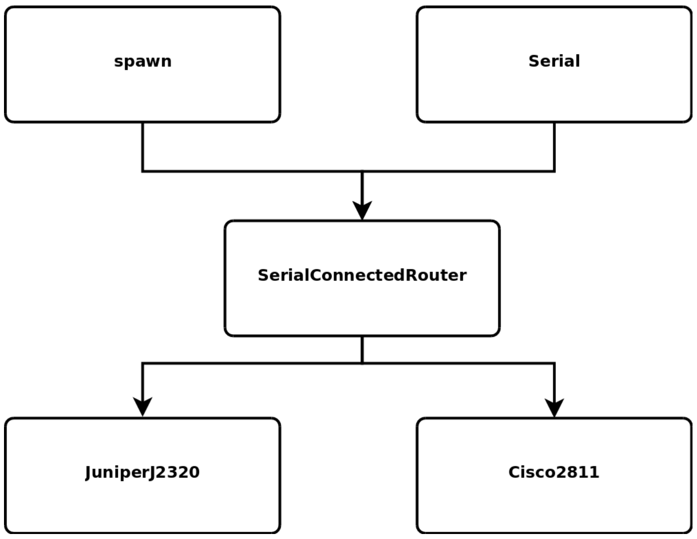

Figure 1: Class derivation diagram.

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 documentation[1] 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 `fdexpect.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 system[2]. 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.:

```
router.setUsername('root')
router.setPassword('secret')
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('delete interfaces ge-0/0/0 unit 0 family inet')
# wait for the configuration prompt
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.:

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

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

[2]The 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.

```
router.gotocli()
router.clipromptexpect()
router.sendcommand("show version")

# jump, in the input stream, after the echo
# of the command that we just sent
router.readuntil('\n')

routerhostname = router.readuntil('\n')
routermodel = router.readuntil('\n')
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 system[3]. 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.:

```
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>')
```

[3]The 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.:

```
# go to configuration mode logging in if needed
router.gotoconfig()
router.sendcommand("interface FastEthernet 0/0")
# wait for the '(config-if)#' prompt
router.config_promptexpect("if")
router.sendcommand("ip address 191.168.0.31 255.255.255.0")
router.config_promptexpect("if")
router.sendcommand("no shutdown")
router.config_promptexpect("if")
router.sendcommand("end")
router.enabledpromptexpect()
router.write()
```

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

```
router.gotocli()
router.clipromptexpect()
router.sendcommand("show version")

# jump, in the input stream, after the echo
# of the command that we just sent
router.readuntil('\n')

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

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
test.addSubtest('initial configuration', task = True)

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

# 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
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.:

```
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.

```python
# ...
import traceback
# ...
# ...

try:
    test.begin(3)

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

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
savedfile = test.save(dir = "/tmp")

# destroy the test object
del test

# 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.:

```
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 cable[5] and an Ethernet cable (§ figure 2).

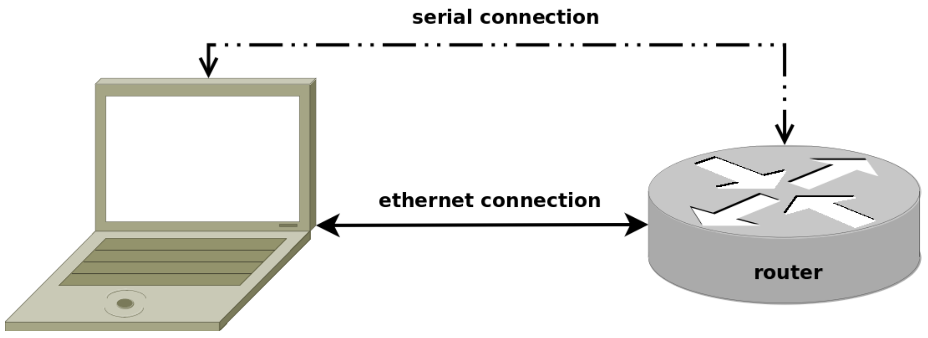

Figure 2: Connections between computer and router equipment for test execution.

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

[5]On 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 refered 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

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"
LOCAL_INTERFACE = 'eth0'
LOCAL_IP = '191.168.0.32'
LOCAL_MASK= '24'
OSPF_AREA = '0.0.0.0'

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

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

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

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

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

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

    test.addSubtest('ospfdisable', task = True)
    test.addSubtestTitle('ospfdisable', "Disable OSPF on the router")
    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)
56      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
    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)
71
        # turn on logging
        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)
81      test.addResult('routerconf', "Router Password", "***")

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

    # 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")

    # 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', "OSPF Area", OSPF_AREA)

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
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:
    test.begin('routerinfo')

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

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

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

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

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

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)
```

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()/"XXXXXXXXXXXXXXXXXX"
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)

177     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:
187     print type(err), err
        test.end('icmp', TEST_FAILED)
    except:
        # Unexpected error
        raise
192 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 corretness is asserted:

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

202     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)
207     pospf.display()

        test.addResult('hello', 'OSPF Type', pospf.type)
        test.addResult('hello', 'OSPF Version', pospf.version)
        test.addResult('hello', 'OSPF Source address', pospf.src)
212     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)
```

Then OSPF is disabled on the router:

```
# now disable ospf on the router
try:
    test.begin('ospfdisable')
    router.gotoconf()
    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
    test.end('ospfdisable', TEST_FAILED)
except:
    # Unexpected error
    raise
else:
    # The test succeeded
    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()

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.

```
# 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:

```
SERIALDEVICE = '/dev/ttyUSB0'
ROUTER_IP = '191.168.0.31'
ROUTER_MASK = '255.255.255.0'
ROUTER_USERNAME = 'admin'
ROUTER_PASSWORD = 'secret'
ROUTER_HOSTNAME = 'cisco2'
ROUTER_INTERFACE = "FastEthernet 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"
```

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 retreving ('routerinfo') subtests:

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

    # 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)

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

    router.gotoconfig()
    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()

    # 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()

    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)

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
else:
    # The test succeeded
    test.end('routerconf', TEST_OK)


# 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)

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)
```

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()

```

```
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 Inital Forming Test and shows the results of its execution with a connected Juniper J2320 router. The integral code listing will be refered to as "listing 11", and can be found in appendix B.2.1.

After the initial import statements, some costants are defined:

```
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	**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 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	**DONE**
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	**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	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.

```
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"
```

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")

test.addSubtest('localconf', task = True)
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
test.addSubtestDependency('routerconf', 'localconf')

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

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

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

    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)
```

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.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_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)

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:

```
# Now begin the formation of an adjacency
#
```

```
#               +----+                                              +----+
#               |ROU|                                               |LOC|
#               +----+                                              +----+
#
#               Down                                                Down
#                                  Hello(DR=0,seen=0)
#                         ------------------------------------>
#                               Hello  (DR=LOC, seen=ROU, ...)      Init
#                         <------------------------------------
#               ExStart          D-D  (Seq=x, I ,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:

```
    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()

    p1[IP].src = LOCAL_IP
    p1[IP].dst = AllSPFRouters

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

    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

    send(p1)
```

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

```
    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)
    assert len(sniffedpackets) > 0
    sniffedpackets.show()
    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

    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)
```

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)



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)
```

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

```
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 *
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 = '255.255.255.0'
ROUTER_USERNAME = 'admin'
ROUTER_PASSWORD = 'secret'
ROUTER_HOSTNAME = 'cisco2'
ROUTER_INTERFACE = "FastEthernet 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"
```

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

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

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

        router.gotoconfig()
        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()


        # 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")
        router.enabledpromptexpect()

        router.gotoconfig()
        router.sendcommand("interface %s" % ROUTER_INTERFACE)
        router.config_promptexpect("if")
        router.sendcommand("ip ospf priority %s" % ROUTER_PRIORITY)
        router.config_promptexpect("if")
        router.sendcommand("end")
        router.enabledpromptexpect()

        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)

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

```
# now disable ospf on the router
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)
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)
```

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:

```
router = Router(JUNIPER_J2320)
router.setUsername('root')
router.setPassword('secret')
router.setConnectionMethod(SERIAL_CONNECTION) # The router is connected to the
                                               # machine running this script
                                               # through a serial port

router.setSerialport('/dev/ttyUSB0')         # The serial port device

router.setIPv4Address(0, '192.168.2.1')      # Interface no. and IP address

router.send()                       # Sync the router's configuration
                                    # with the Router object

# Perform some tests using packet forging
# ...
# ...

router.retrieve()                   # Retrieve the state of the router

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 protocols[6]. 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 partecipate 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():
    # ...
    # ...
    def execute(self):
        "This method should be overridden by derived classes."
        pass

    def run(self):
        "Encapsulates the call to execute()"
        try:
            self.checkForDependencies()  # method to be defined too
            self.execute()
        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
```

[6]Note 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`:

```
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
# 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

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

class SerialConnectedRouter(serial.Serial, fdpexpect.fdspawn):
    """
    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)

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

        # Turn off buffering
        self.maxread=1

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

    def sendline(self, line):
        "Send a line to the router. Overrides fdpexpect.spawn.sendline()"
        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."""
        print "Sending: %s" % command
        for c in command:
            self.send(c)
            time.sleep(COMMANDSLOWNESS)
        self.sendline("")
        time.sleep(SLEEPAFTERWRITE)
        self.flush()
        self.flushOutput()

    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.")

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

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

A.2 The juniperj2320 module

Listing 5: The juniperj2320 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.

from serialrouter import *
import fdpexpect
import time

TIMEOUT = 4
BAUDRATE = 9600
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."

class JuniperJ2320(SerialConnectedRouter):
    """
    This class represents a Juniper J2320 Router connected via
    serial port.
    """
    def __init__(self, serialdevice):
        self.loginprompt = "ogin:"
        self.initprompt = "%"
        self.cliprompt = "\>"
        self.confprompt = "\#"
        self.username = None
        self.password = None
        SerialConnectedRouter.__init__(self, serialdevice, BAUDRATE, \
                BYTESIZE, PARITY, STOPBITS, TIMEOUT)

    def setUsername(self, username):
        self.username = username
        self.initprompt = "%s@.*\%%" % self.username
        self.cliprompt = "%s@.*\>" % self.username
        self.confprompt = "%s@.*\#" % self.username
        self.promptlist = [self.loginprompt, self.initprompt,
                self.cliprompt, self.confprompt,
                fdpexpect.TIMEOUT, fdpexpect.EOF]

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

    def gotologinscreen(self, sendnewline = True):
        """
        Climb the JunOS configuration hierarchy
        until the login prompt appears
        """
        if sendnewline:
            self.sendline("")
        i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
        if i == 0: # loginprompt
            pass # success
        # initprompt, cliprompt, confprompt
        elif i == 1 or i == 2 or i == 3:
            self.sendcommand("exit")
            self.gotologinscreen(sendnewline=False)
        elif i == 4: # timeout
            raise RouterConfigurationException(TIMEOUT_ERROR)
        elif i == 5: # eof
            raise RouterConfigurationException(EOF_ERROR)

    def login(self):
        if self.username == None or self.password == None:
            raise RouterConfigurationException("Username or password not set."
                )
        self.sendline("")
        i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
        if i != 0:
            self.gotologinscreen()
        self.sendcommand(self.username)
        self.timedexpect("ssword:", timeout = 12)
        self.sendcommand(self.password)
        self.initpromptexpect(timeout = 20)

    def gotocli(self, sendnewline = True):
        if sendnewline:
            self.sendline("")
        i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
        if i == 0: # loginprompt
            self.login()
            self.gotocli(sendnewline = False)
            return
        elif i == 1: # initprompt
            self.sendcommand("cli")
            self.gotocli(sendnewline = False)
            return
        elif i == 2: # cliprompt
            pass # success
        elif i == 3: # confprompt
            self.sendcommand("exit")
            self.gotocli(sendnewline = False)
            return
        elif i == 4: # timeout
            raise RouterConfigurationException(TIMEOUT_ERROR)
        elif i == 5: # eof
            raise RouterConfigurationException(EOF_ERROR)
        self.sendcommand("set cli screen-length 0")
        self.timedexpect("length set to 0")
        self.clipromptexpect()

    def gotoconf(self, sendnewline = True):
        if sendnewline:
            self.sendline("")
        i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
        if i == 0: # loginprompt
            self.login()
            self.gotoconf(sendnewline = True)
        elif i == 1: # initprompt
            self.gotocli()
            self.gotoconf(sendnewline = True)
        elif i == 2: # cliprompt
            self.sendcommand("configure")
            self.gotoconf(sendnewline = False)
        elif i == 3: # confprompt
            pass # success
        elif i == 4: #timeout
            raise RouterConfigurationException(TIMEOUT_ERROR)
        elif i == 5:
            raise RouterConfigurationException(EOF_ERROR)

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

    def confpromptexpect(self, timeout=DEFAULTTIMEOUT):
        try:
            self.timedexpect(self.confprompt, timeout)
        except RouterConfigurationException:
            self.sendline("")
            self.timedexpect(self.confprompt, timeout)

    def clipromptexpect(self, timeout=DEFAULTTIMEOUT):
        try:
            self.timedexpect(self.cliprompt, timeout)
        except RouterConfigurationException:
            self.sendline("")
            self.timedexpect(self.cliprompt, timeout)

    def initpromptexpect(self, timeout=DEFAULTTIMEOUT):
        try:
            self.timedexpect(self.initprompt, timeout)
        except RouterConfigurationException:
            self.sendline("")
            self.timedexpect(self.initprompt, timeout)
```

A.3 The cisco2811 module

Listing 6: The cisco2811 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.

from serialrouter import *
import fdpexpect
import time

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

class Cisco2811(SerialConnectedRouter):
    """
    This class represent a Cisco 2811 Router connected via
    serial port.
    """
    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)
    def setUsername(self, username):
        self.username = username

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

    def setHostname(self, hostname):
        self.hostname = hostname
        self.enabledprompt = "%s#" % hostname
        self.cliprompt = "%s>" % hostname
        self.configprompt = "%s.config.#" % hostname
        self.config_prompt = "%s.config-.*#" % hostname
        self.promptlist = [self.loginprompt, self.cliprompt, self.
            enabledprompt, \
                self.passwordprompt, self.configprompt, self.config_prompt, \
                fdpexpect.TIMEOUT, fdpexpect.EOF]

    def gotologinscreen(self):
        """
        Climb the IOS configuration hierarchy
        until the login prompt appears
        """
        self.sendcommand("")
        i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
        if i == 0: # loginprompt
            pass    # success
       # cliprompt, enabledprompt, configprompt, config_prompt
        elif i == 1 or i == 2 or i == 4 or i == 5:
            self.sendcommand("exit")
            time.sleep(ADDITIONALSLEEP)
            self.gotologinscreen()
        elif i == 3: # passwordprompt
            self.sendcommand("")
            self.gotologinscreen()
        elif i == 6: #timeout
            if self.tries == 0:
                self.sendcommand("")
                self.tries += 1
            else:
                self.tries = 0
                raise RouterConfigurationException(TIMEOUT_ERROR)
        elif i == 7:
            raise RouterConfigurationException(EOF_ERROR)

    def login(self):
        if self.username == None or self.password == None or self.hostname ==
            None:
            raise RouterConfigurationException("Username or password or
                hostname not set.")
        time.sleep(ADDITIONALSLEEP)
        i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
        if i != 0:
            self.gotologinscreen()
            self.sendcommand("")
            self.login()
            return
#        self.timedexpect(self.loginprompt)
        self.sendcommand(self.username)
        self.timedexpect(self.passwordprompt, timeout = 12)
        self.sendcommand(self.password)
        self.clipromptexpect(timeout = 20)
        self.sendcommand("terminal length 0")
        self.clipromptexpect()

    def gotocli(self, sendnewline = True):
        if sendnewline:
            self.sendcommand("")
        i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
        if i == 0: # loginprompt
            self.login()
            self.gotocli()
            return
        elif i == 1: # cliprompt: bingo
            pass
        elif i == 2 or i == 4 or i == 5: # enabledprompt, configprompt, config_prompt
            self.sendcommand("exit")
            time.sleep(ADDITIONALSLEEP)
            self.gotocli(sendnewline = True)
            return
        elif i == 3: # passwordprompt
            self.sendcommand("")
            self.gotocli(sendnewline = False)
            return
        elif i == 6: # timeout
            raise RouterConfigurationException(TIMEOUT_ERROR)
        elif i == 7:
            raise RouterConfigurationException(EOF_ERROR)
        self.sendcommand("terminal length 0")
        self.clipromptexpect()

    def gotoenabled(self, sendnewline = True):
        if sendnewline:
            self.sendcommand("")
        i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
        if i == 0: # loginprompt
            self.login()
            self.gotoenabled()
        elif i == 1: # cliprompt
            self.sendcommand("enable")
            time.sleep(ADDITIONALSLEEP)
            self.gotoenabled(sendnewline = False)
        elif i == 2: # enabledprompt: bingo
            pass
        elif i == 3: # passwordprompt
            self.sendcommand(self.password)
            self.gotoenabled(sendnewline = False)
        elif i == 4 or i == 5: # configprompt, config_prompt
            self.sendcommand("exit")
            time.sleep(ADDITIONALSLEEP)
            self.gotoenabled(sendnewline = True)
        elif i == 6: # timeout
            raise RouterConfigurationException(TIMEOUT_ERROR)
        elif i == 7:
            raise RouterConfigurationException(EOF_ERROR)

    def gotoconfig(self, sendnewline = True):
        if sendnewline:
            self.sendcommand("")
        i = self.listexpect(self.promptlist, timeout=DEFAULTTIMEOUT)
        if i == 0: # loginprompt
            self.login()
            self.gotoenabled()
            self.gotoconfig()
        if i == 1: # cliprompt
            self.gotoenabled()
            self.gotoconfig()
        elif i == 2: # enabledprompt
            self.sendcommand("configure terminal")
            self.gotoconfig(sendnewline = False)
        elif i == 3: # passwordprompt
            self.sendcommand("")
            self.gotoconfig(sendnewline = False)
        elif i == 4: # configprompt: bingo
            pass
        elif i == 5: # config_prompt
            self.sendcommand("exit")
            time.sleep(ADDITIONALSLEEP)
            self.gotoconfig(sendnewline = True)
        elif i == 6: # timeout
            raise RouterConfigurationException(TIMEOUT_ERROR)
        elif i == 7:
            raise RouterConfigurationException(EOF_ERROR)

    def write(self):
        self.gotoenabled()
        self.sendcommand("write")
        self.timedexpect("[OK]", timeout=20)
    def enabledpromptexpect(self, timeout=DEFAULTTIMEOUT):
        try:
            self.timedexpect(self.enabledprompt, timeout)
        except RouterConfigurationException:
            self.sendcommand("")
            self.timedexpect(self.enabledprompt, timeout)

    def configpromptexpect(self, timeout=DEFAULTTIMEOUT):
        try:
            self.timedexpect(self.configprompt, timeout)
        except RouterConfigurationException:
            self.sendcommand("")
            self.timedexpect(self.configprompt, timeout)

    def config_promptexpect(self, subprompt=".*", timeout=DEFAULTTIMEOUT):
        config_p = "%s.config-%s.#" % (self.hostname, subprompt)
        try:
            self.timedexpect(config_p, timeout)
        except RouterConfigurationException:
            self.sendcommand("")
            self.timedexpect(config_p, timeout)

    def clipromptexpect(self, timeout=DEFAULTTIMEOUT):
        try:
            self.timedexpect(self.cliprompt, timeout)
        except RouterConfigurationException:
            self.sendcommand("")
            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:

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

    # Initial setup
    test.addSubtest('SetUp', task=True)
    test.addSubtestTitle('SetUp', "Initial configuration")
    try:
        test.begin('SetUp')
        # Perform set-up operations
        # ...
    except:
        test.end('SetUp', TEST_FAILED)
    else:
        # The setup succeeded
        test.end('SetUp', TEST_OK)


    # First sub-test
    test.addSubtest('FirstTest') # labels may be strings or numbers
    test.addSubtestTitle('FirstTest', "Just a test")
    test.addSubtestDependency('FirstTest', 'SetUp') # 'FirstTest' depends on '
         SetUp'
    # begin the test
    try:
        test.begin('FirstTest')
        # Perform the sub-test
        # ...
        # Do some assertions
        # assert(...)
        # ...
        # Collect some results during the sub-test
        test.addResult('FirstTest', 'HelloInterval', 10)
        # ...
        test.addResult('FirstTest', 'DeadInterval', 40)
        # ...
        test.addResult('FirstTest', 'Flags', 'E+M')
    except TestDependencyException:
        # The dependencies were not met
        test.end('FirstTest', TEST_SKIPPED)
    except Exception, err:
        # An error occurred
        print type(err), err
        test.end('FirstTest', TEST_FAILED)
    except:
        # Unexpected error
        raise
    else:
        # The test succeeded
        test.end('FirstTest', TEST_OK)

    # Second sub-test
    test.addSubtest(2)
    test.addSubtestTitle(2, "Another test")
    # begin the test
    try:
        test.begin(2)
        # Perform the sub-test
        # ...
    except Exception, err:
        # An error occurred
        print type(err), err
        test.end(2, TEST_FAILED)
    except:
        # Unexpected error
        raise
    else:
        # The test succeeded
        test.end(2, TEST_OK)

    # Output the summary on the screen
    print test

    # Save the test summary
    test.save()

"""

import os

TEXTWIDTH = 80
DESCFIELD = 0.6 # percentage of TEXTWIDTH
SEPFIELD = 0.1  # percentage of TEXTWIDTH
VALFIELD = 0.3  # percentage of TEXTWIDTH

TEST_FAILED = 0
TEST_OK = 1
TEST_SKIPPED = 2

# Find ansi color codes
try:
    import curses
except:
    colorcodes = {
            "black": "",
            "red": "",
            "green": "",
            "yellow": "",
            "blue": "",
            "magenta": "",
            "cyan": "",
            "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)
            }

import pickle
import time

class TestException(Exception):
    pass

class TestDependencyException(TestException):
    pass

class InvalidCommandException(TestException):
    pass


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

    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"
            else:
                val = "Failed"
        else:
            val = self.value
        res += "%*s" % (descwidth, self.description)
        res += " " * sepwidth
        res += "%-*s" % (valwidth, val)
        res += "\n"
        return res

    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



class SubTest():
    """
    This class represents a subtest. Each subtest is made of
    various results.
    """
    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

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

    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

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

    def getFinalResultString(self):
        """
        Returns a string representation of the final result
        of the subtest (e.g. "PASSED" or "FAILED" or "DONE")
        """
        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"

    def getFinalResultColorString(self):
        """
        Like getFinalResultString but using ANSI colors.
        """
        res = ""
        if self.finalresult == TEST_OK:
            res += colorcodes["green"]
        else:
            res += colorcodes["red"]
        res += self.getFinalResultString()
        res += colorcodes["white"]
        return res

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

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

    def __str__(self):
        descwidth = int(TEXTWIDTH * DESCFIELD)
        sepwidth = int(TEXTWIDTH * SEPFIELD)
        valwidth = int(TEXTWIDTH * VALFIELD)
        res = ""
        res += colorcodes["white"]
        res += "=" * TEXTWIDTH
        res += "\n"
        res += colorcodes["yellow"]
        res += "%*s" % (descwidth, self.title)
        res += colorcodes["white"]
        res += " " * (sepwidth)
        res += "%-*s" % (valwidth, self.getFinalResultColorString())
        res += "\n"

        if self.results:
            res += "-" * TEXTWIDTH
            res += "\n"

        for result in self.results:
            res += str(result)

        return res

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

        if self.results:
            res += "\t\\hline \n"

        for result in self.results:
            res += result.getTeX()

        return res



class Test():
    """
    This class represents a test. Each test is made of various
    subtests. Each subtest is made of various results.
    """
    def __init__(self, test_title):
        """
        Initializes a Test instance.
        test_title should be a string with the title of the test.
        """
        self.title = test_title
        self.subtests = dict()
        self.currentindex = 0
        self.subtestorder = dict()
        self.dependencies = dict()

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

    def addSubtest(self, subtestlabel, task = False):
        """
        Add a subtest with label subtestlabel, which can be of
        any immutable type (int, string, ...)
        If task == True, the subtest is considered a task,
        i.e. prints "DONE" or "ERROR" instead of "PASSED"
        or "FAILED"
        """
        # We have a dictionary label -> subtest object
        # and a dictionary index -> label
        # to preserve the order of the subtests

        newsubtest = SubTest(subtestlabel, task)
        self.subtests.update({subtestlabel: newsubtest})
        self.subtestorder.update({self.currentindex: subtestlabel})
        self.currentindex += 1

    def addSubtestDependency(self, subtestlabel, dependsonlabel):
        """
        Specifies that the subtest specified by subtestlabel depends
        on the success of the subtest specified by dependsonlabel.
        """
        if not self.subtests.has_key(subtestlabel):
            errstr = "Invoke addSubTest() first"
            raise InvalidCommandException, errstr
        testdeps = self.dependencies.get(subtestlabel, [])
        testdeps.append(dependsonlabel)
        self.dependencies.update({subtestlabel: testdeps})

    def addSubtestTitle(self, subtestlabel, subtesttitle):
        "Give a title to the subtest"
        if not self.subtests.has_key(subtestlabel):
            self.addSubtest(subtestlabel)
        self.subtests[subtestlabel].setTitle(subtesttitle)

    def printTitleString(self, outstring):
        "Used to print fancy messages on the screen"
        print colorcodes["blue"], "-" * TEXTWIDTH
        print "\t", outstring
        print colorcodes["blue"], "-" * TEXTWIDTH, colorcodes["white"]

    def announce(self, outstring):
        "Used to print short messages on the screen"
        print ""
        print colorcodes["yellow"], "\t", outstring, colorcodes["white"]
        print ""

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

        self.subtests[subtestlabel].setFinalResult(failure_result_value)

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

        # check dependencies
        for dep in testdeps:
            try:
                dept = self.subtests[dep]
            except KeyError:
                errstr = "Test %s not found" % dep
                raise InvalidCommandException, errstr

            # if the test on which the current test depends
            # were not passed, raise an exception
            if dept.getFinalResult() != TEST_OK:
                outstring = "%sSkip: %s%s.%s"   \
                        % (colorcodes["yellow"], \
                          colorcodes["magenta"], \
                          self.subtests[subtestlabel].getTitle(), \
                          colorcodes["white"])
                self.printTitleString(outstring)
                errstr = "Dependencies not met for test %s." % subtestlabel
                raise TestDependencyException, errstr

        outstring = "%sBegin: %s%s.%s" % (colorcodes["yellow"], \
                colorcodes["magenta"], \
                self.subtests[subtestlabel].getTitle(), \
                colorcodes["white"])
        self.printTitleString(outstring)

    def addResult(self, subtestlabel, description, result):
        "Add a result to a sub-test"
        self.subtests[subtestlabel].addResult(description, result)

    def end(self, subtestlabel, finalresult = False):
        """
        Finalize the subtest, specifying the final result
        (e.g. TEST_OK or TEST_FAILED)
        """
        try:
            self.subtests[subtestlabel].setFinalResult(finalresult)
        except KeyError:
            errstr = "Test %s not found" % subtestlabel
            raise InvalidCommandException, errstr

        if finalresult == TEST_SKIPPED:
            return
        elif finalresult == TEST_OK:
            resstr = colorcodes["green"]
        else:
            resstr = colorcodes["red"]


        resstr += self.subtests[subtestlabel].getFinalResultString()
        resstr += colorcodes["white"]
        outstring = "%sFinish: %s%s.%s Result: %s%s" % (colorcodes["yellow"],
            \
                colorcodes["magenta"], \
                self.subtests[subtestlabel].getTitle(), \
                colorcodes["white"], \
                resstr, \
                colorcodes["white"])
        self.printTitleString(outstring)

    def __str__(self):
        "Returns the summary of the test"
        title = ""
        title += "   " + colorcodes["white"]
        title += self.title
        title += "   " + colorcodes["blue"]
        res = ""
        res += colorcodes["blue"]
        res += "*" * TEXTWIDTH
        res += "\n"
        res += title.center( \
                TEXTWIDTH + \
                len(colorcodes["white"]) + \
                len(colorcodes["blue"]), \
                "*")
        res += "\n"
        res += "*" * TEXTWIDTH
        res += "\n"
        res += colorcodes["white"]
        orderkeys = self.subtestorder.keys()
        orderkeys.sort()
        for key in orderkeys:
            label = self.subtestorder[key]
            subtest = self.subtests[label]
            res += str(subtest)

        res += "=" * TEXTWIDTH
        res += "\n"
        res += colorcodes["blue"]
        res += "*" * TEXTWIDTH
        res += "\n"
        res += colorcodes["white"]
        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 += "\t\\hline \n"
        res += "\t\\multicolumn{2}{|c|}{%s} \\\\ \n" % title
        res += "\t\\hline \n"

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

        res += "\t\\hline \n"
        res += "\\end{tabular}"
        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

        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
            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()
    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"
```

```
if __name__ == "__main__":
    # A new Test() instance
    test = Test("Foo Bar")

    # Initial setup
    test.addSubtest('SetUp', task=True)
    test.addSubtestTitle('SetUp', "Initial configuration")
    test.begin('SetUp')
    test.end('SetUp', TEST_OK)

    # First sub-test
    test.addSubtest('FirstTest') # labels may be strings or numbers
    test.addSubtestTitle('FirstTest', "Just a test")
    test.addSubtestDependency('FirstTest', 'SetUp') # 'FirstTest' depends on '
        SetUp'
    # begin the test
    try:
        test.begin('FirstTest')
        # Collect some results during the sub-test
        test.addResult('FirstTest', 'HelloInterval', 10)
        test.addResult('FirstTest', 'DeadInterval', 40)
        test.addResult('FirstTest', 'Flags', 'E+M')
    except TestDependencyException:
        # The dependencies were not met
        test.end('FirstTest', TEST_SKIPPED)
    except Exception, err:
        # An error occurred
        print type(err), err
        test.end('FirstTest', TEST_FAILED)
    except:
        # Unexpected error
        raise
    else:
        # The test succeeded
        test.end('FirstTest', TEST_OK)

    # Second sub-test
    test.addSubtest(2)
    test.addSubtestTitle(2, "Another test")
    # begin the test
    test.begin(2)
    test.end(2, TEST_FAILED)
#   test.end(2, TEST_OK)

    # Third sub-test
    test.addSubtestTitle('3', "And another one")
    test.addSubtestDependency('3', 2)
    # begin the test
    try:
        test.begin('3')
        # Collect some results
        test.addResult('3', 'Supercapsule', "OK!")
    except TestDependencyException:
        # The dependencies were not met
        test.end('3', TEST_SKIPPED)
    except Exception, err:
        # An error occurred
        print type(err), err
        test.end('3', TEST_FAILED)
    except:
        # Unexpected error
        raise
    else:
        # Test succeeded
        test.end('3', TEST_OK)

    # Output on the screen
    print test
    print test.getTeX()

    # Output on a file
    savedfile = test.save()
    # Load from file
    del test
    test = testload(savedfile)
    print test

    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

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

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"

test = Test("Juniper J2320 Basic Test")

# The subtests
test.addSubtest('localconf', task = True)
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
test.addSubtestDependency('routerconf', 'localconf')

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:
    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)

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

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

    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))

    # 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")
    # 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', "OSPF Area", OSPF_AREA)

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
else:
    # The test succeeded
    test.end('routerconf', TEST_OK)


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

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

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

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

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

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

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)


# now check connectivity using icmp
try:
    test.begin('icmp')
    test.announce("Checking connectivity using ICMP")

    conf.iface = LOCAL_INTERFACE

    # an icmp echo-request packet
    icmp_echo_request = IP(dst=ROUTER_IP)/ICMP()/"XXXXXXXXXXXXXXXXX"

    print "Sending an ICMP echo-request packet"

    assert(icmp_echo_request != None)
    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)

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...")

    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()

    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.gotoconf()
```

```
        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
        test.end('ospfdisable', TEST_FAILED)
    except:
        # Unexpected error
        raise
    else:
        # The test succeeded
        test.end('ospfdisable', TEST_OK)

# turn off logging
logfile.close()

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

B.1.2 The Basic Test for Cisco 2811

Listing 10: The basic test script for Cisco 2811.

```
# 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

SERIALDEVICE = '/dev/ttyUSB0'
ROUTER_IP = '191.168.0.31'
ROUTER_MASK = '255.255.255.0'
ROUTER_USERNAME = 'admin'
ROUTER_PASSWORD = 'secret'
ROUTER_HOSTNAME = 'cisco2'
ROUTER_INTERFACE = "FastEthernet 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"

test = Test("Cisco 2811 Basic Test")

# The subtests
test.addSubtest('localconf', task = True)
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
test.addSubtestDependency('routerconf', 'localconf')

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:
    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)

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

    # 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)

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

    router.gotoconfig()
    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()

    # 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()

    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)

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
else:
    # The test succeeded
    test.end('routerconf', TEST_OK)


# 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)

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)


# now check connectivity using icmp
try:
    test.begin('icmp')
    test.announce("Checking connectivity using ICMP")

    conf.iface = LOCAL_INTERFACE

    # an icmp echo-request packet
    icmp_echo_request = IP(dst=ROUTER_IP)/ICMP()/"XXXXXXXXXXXXXXXXXX"

    print "Sending an ICMP echo-request packet"

    assert(icmp_echo_request != None)

    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)

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...")

    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()

    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
```

```
        raise
    else:
        # The test succeeded
        test.end('ospfdisable', TEST_OK)

# 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"

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

test.addSubtest('localconf', task = True)
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
test.addSubtestDependency('routerconf', 'localconf')

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

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

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_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)

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)


# Now begin the formation of an adjacency
#
#           +---+                                         +---+
#           |ROU|                                         |LOC|
#           +---+                                         +---+
#
#           Down                                          Down
#                        Hello(DR=0,seen=0)
#                   ------------------------------>
#                     Hello (DR=LOC,seen=ROU,...)          Init
#                   <------------------------------
#           ExStart     D-D (Seq=x,I,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

    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()

    p1[IP].src = LOCAL_IP
    p1[IP].dst = AllSPFRouters

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

    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

    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)
    assert len(sniffedpackets) > 0
    sniffedpackets.show()
    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

    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
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)




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)
```

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

# 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 = '255.255.255.0'
ROUTER_USERNAME = 'admin'
ROUTER_PASSWORD = 'secret'
ROUTER_HOSTNAME = 'cisco2'
ROUTER_INTERFACE = "FastEthernet 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"

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

test.addSubtest('localconf', task = True)
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
test.addSubtestDependency('routerconf', 'localconf')

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

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

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 = Cisco2811(SERIALDEVICE)
    router.setUsername(ROUTER_USERNAME)
    router.setPassword(ROUTER_PASSWORD)
    router.setHostname(ROUTER_HOSTNAME)

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

    router.gotoconfig()
    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()


    # 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")
    router.enabledpromptexpect()

    router.gotoconfig()
    router.sendcommand("interface %s" % ROUTER_INTERFACE)
    router.config_promptexpect("if")
    router.sendcommand("ip ospf priority %s" % ROUTER_PRIORITY)
    router.config_promptexpect("if")
    router.sendcommand("end")
    router.enabledpromptexpect()

    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)

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)


# Now begin the formation of an adjacency
#
#             +----+                                            +----+
#             |ROU|                                             |LOC|
#             +----+                                            +----+
#
#             Down                                              Down
#                            Hello(DR=0,seen=0)
#                       ------------------------------>
#
#                         Hello  (DR=LOC, seen=ROU, ...)        Init
#                       <------------------------------
#
#             ExStart       D-D  (Seq=x, I, 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

    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()

    p1[IP].src = LOCAL_IP
    p1[IP].dst = AllSPFRouters

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

    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

    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)
    assert len(sniffedpackets) > 0
    sniffedpackets.show()
    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

    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
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)
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)




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

List of Tables

List of Figures

Copyright