Towards Modeling and Testing of IP Routing Protocols

Jianping Wu, Zhongjie Li, and Xia Yin

Department of Computer Science and Technology, Tsinghua University
Beijing 100084, P.R.China
jianping@cernet.edu.cn
{lzj,yxia}@csnet1.cs.tsinghua.edu.cn

Abstract. Routing protocols are typical distributed systems characterized by dynamic, concurrent and distributed behaviors. As the primary function of routing protocols, routing information processing constitutes the main content of routing protocol conformance testing. However, there isn't a clear model based on which convincing test architecture and test notation can be designed. Also, some important features of routing protocols have not been considered sufficiently in existing test practice. On the other hand, generalized distributed system models, test architectures and test notations usually have limitations when applied directly to specific protocol testing. So it is necessary to study specific cases in order to find a pragmatic and efficient approach. In this paper, an MP-FSM (Finite State Machine with Multiple Ports) model is proposed to describe routing information processing of IP routing protocols. Based on this model and other test requirement particularities, a test architecture called PADTACC (Parallel And Distributed Test Architecture with Centralized-Control) is presented and a test notation called RIPTS (Routing Information Processing Test Script) is defined. We implement the whole approach into a software tester – IRIT (IP Routing Information Tester).

1 Introduction

Routing protocols are core components of the Internet architecture. Therein, the correct running of routing protocols enables connectivity, stability and security of the Internet. In this work we study conformance testing of Internet IP routing protocols including RIP, OSPF and BGP [1].

Inside a router, routing protocols are used to update the routing table dynamically. The main functions of a routing protocol can be divided into two parts: network communication (NC) and routing information processing (RI-Pro). The NC part usually appears in an FSM form functioning to handle low-layer network accessing, establish reliable communication channels for the routing information flows, or detect changes of local network connections. For example, RIP uses UDP to carry its protocol messages. There is no state machine in RIP. OSPF runs on IP, and defines two state machines: the interface machine and the neighbor machine mainly used for Designated Router election, neighboring adjacency establishment and maintenance. BGP has a state machine specifying the procedure how the peers set up, maintain the TCP connections and negotiate parameters. The NC part is relatively easy to model,

D. Hogrefe and A. Wiles (Eds.): TestCom 2003, LNCS 2644, pp. 49–62, 2003.

implement and test. There have been many researches in aspects of protocol modeling, test generation, test system construction, etc [2-5]. Most of them use TTCN (Tree and Tabular Combined Notation) [6] as the test notation to specify an abstract test suite.

The RI-Pro part consists of routing information origination and propagation as well as routing table calculation and update. This part represents the primary function of a routing protocol and should also be the main content of routing protocol conformance testing. There is relatively less work on this part. In reference [5], the authors find that traditional means of testing communication protocols are inappropriate for the testing of routing protocols in the aspects of the test architecture, test notation, etc. They propose some enhancing techniques to improve the test system. SOCRATES [7] is a software test tool exploring automatic network topology generation and probabilistic algorithms, it concentrates on the testing of routing information processing, but it is incapable of testing the inter-area and AS-external routing behaviors of OSPF. As to test systems, currently there are some commercial products, such as HP RouterTester [8], NetCom Systems Smartbits [9], etc. They can be used for both conformance and stress testing of IP routing protocols, focusing on the routing information processing part. They all use self-defined test notation to write test specifications.

Despite these works, there isn't a clear model for routing information processing based on which convincing test architecture and test notation can be designed. Also, some important features of routing protocols have not been considered sufficiently. Some of those features are:

- routing information grouping. Pieces of routing information are packed in one or several packets and come out of the IUT (Implementation Under Test: a software module or system which implements the tested protocol) in a random order. The pieces of routing information contained in one packet appear in a random order too. A general test language like TTCN is clumsy to manipulate these ordering variations. An intelligent "multiple-packets and set matching" mechanism is appropriate.
- complex test configurations. Test configuration includes configuration of the tester and the IUT. Routing protocols such as OSPF have many tunable parameters, options and configuration scenarios. Furthermore, it is common that one test case uses several test configurations. This situation necessitates formalization and automation of test configuration operations.
- complex protocol behaviors. Complex behaviors in routing protocols are difficult to describe in a general test language like TTCN. It is better to program such behaviors by C language. While testing routing information processing, we assume that the network communication part has been tested and is correct. If this part is complex, we'd better implement it by a simulated router which could communicate with the IUT like a real router.

In this paper, we will introduce a formal model of the routing information processing (sect. 2), analyze the specific requirements of routing protocol conformance testing, design an applicable test architecture (Sect. 3) and a flexible test notation and discuss the efficient means of test organization (sect. 4). The complete approach is

implemented as a linux-based software tester called IRIT – IP Routing Information Tester (sect. 5). Sect. 6 gives example test topologies of OSPF protocol. Sect. 7 concludes the paper with some discussion on future work.

2 The MP-FSM Model

Fig. 1 shows the skeleton of a routing protocol and its relation with other modules in a router. The NC part lies in the bottom serving the RI-Pro part with low-layer network accessing, reliable communication channels or local connection monitoring. The RI-Pro part is responsible for originating routing information pieces describing local network interfaces and for flooding of the information generated by itself and by others to its neighboring routers (using routing information origination and flooding algorithms). All routing information collected by a router composes its Routing Information Base (RIB). The internal routing table (I-RT) is calculated based on the RIB (using routing table calculation algorithm). In addition, another routing protocol may run on the same router (AS boundary router). Routing management module provides interfaces for routing protocols to modify the system routing table (S-RT). It also manages the route redistribution between different routing protocols. Users configure or monitor the routing protocols via the CLI (Command Line Interface) provided by OAM (Operation and Management). MIBs (Management Information Base) of routing protocols are managed by SNMP module. Both the CLI and MIBs can be used to examine the RIB, I-RT, running statistics and various internal states.

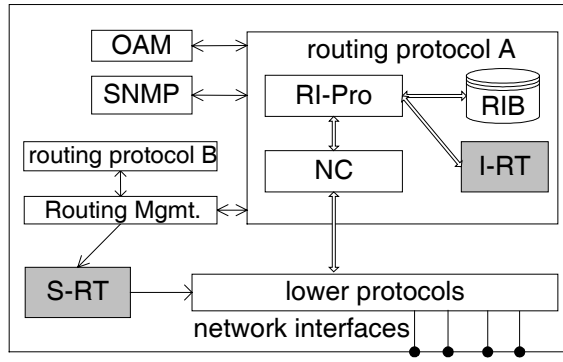

Fig. 1. Routing Protocols in a router

The routing information processing part of routing protocols is described in natural language and there is no clear functional or procedural model. We first give a formal model in definition 1. It is used for later discussion of the test architecture and test notation design.

Definition 1 (MP-FSM model of routing information processing) The routing information processing of IP routing protocols (under a given protocol configuration)

can be modeled as an FSM with multiple ports which is defined as an 8-tuple M= $<S, I, O, \sum, P, f, g, s_0>$ where

S is the set of states of the routing information processing procedure. A state is defined by the combination of RIB and I-RT: $<R, T>$. S is finite in practice under a given network topology and protocol configuration. s_0 is the initial state with a RIB and I-RT which are generated by this router at startup.

$\sum = L \cup E$, is a collection of observable routing information between this router and the environment (neighboring routers). The messages include internal routing information set L and external routing information set E.

P is the set of router's ports. A port is an abstraction of an interface through which all the routing information is communicated among routers. $P=\{1,2,..,r\}$ is the port index. All ports are bidirectional.

I is the set of input actions, $I \subseteq P \times \{recv\} \times \sum$, an input action is an input of routing information from a specific port.

O is the set of output actions, $O \subseteq P \times \{send\} \times \sum$, an output action is an output of routing information to a specific port.

f is the output function, $S \times I \rightarrow P(O)$, where $P(O)$ is the power set of O. f is the routing information origination and flooding algorithms. The output actions are parallel if they occur in different ports, serial if they occur in the same port. Their sequencing can be arbitrary in either case.

g is the transition function, $S \times I \rightarrow S$. g is the RIB update and routing table calculation algorithm.

A transition can be represented as $s_i \xrightarrow{i/\Omega} s_j$ which states that an MP-FSM, in state s_i, upon an input action i, moves to state s_j and produces a set of output actions Ω ($\subseteq O$). Following this notation we define $f(s_0, i_1 i_2 .. i_n) = \Omega_1 \Omega_2 .. \Omega_n$, $g(s_0, i_1 i_2 .. i_n) = s_n$, iff. $f(s_{k-1}, i_k) = \Omega_k$, $g(s_{k-1}, i_k) = s_k$, $1 \le k \le n$. This is used to describe consecutive input and output actions. Fig. 2 illustrates the MP-FSM model and a state transition scenario where the machine moves from state s_1 to s_2 with input ***a*** at port p1, output ***b*** at port p2 and output ***c*** at port p3 respectively.

3 The PADTACC Test Architecture

Test architecture describes the environment the IUT lies in, generally the connection relations between the IUT and the test system. The behaviors of routing protocols are distributed at several router interfaces interacting with the test system concurrently. Therefore, what we need is a test architecture that supports multiple test points and multiple test processes. This idea immediately results in a simple but widely applied test architecture [10] in which several testers surround the IUT and compose a test system (shown in Fig.3). TTCN-2 [6] improves this test architecture. It introduces the concept of test component (TC) which runs a test process independently. In TTCN-2 test architecture (see Fig.4), there is one Main Test Component (MTC) which creates other test components named Parallel Test Component (PTC). The communication

interface between test components is called Coordination Point (CP) and between test component and the IUT Point of Control and Observation (PCO). CP and PCO are both modeled as two FIFO queues: one for each direction of communication. PCOs are connected to the ports of the IUT thru the underlying networks.

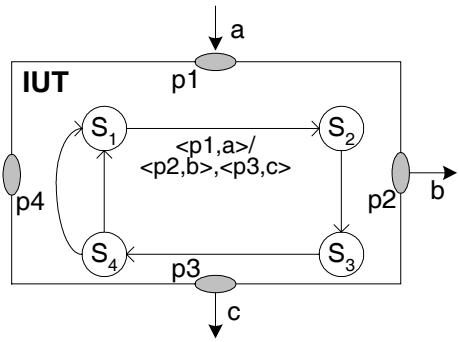

Fig. 2. The MP-FSM Model

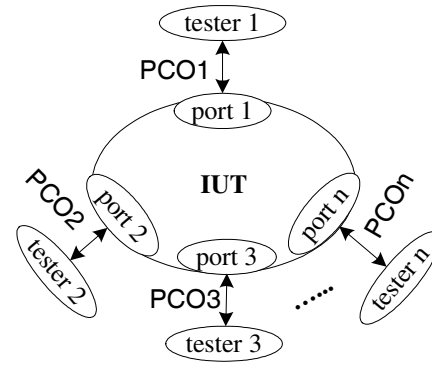

Fig. 3. A General Distributed Test Architecture

This test architecture is sufficient for use but too generalized. Some modifications can provide a more practical variation. First, we simplify this test architecture by removing CPs between test components. Synchronizations between PTCs will be handled by MTC. As we'll see later, no problems or overhead occur because the necessary synchronizations between PTCs are very simple. Second, we lighten the MTC by cutting its connections to the IUT. Third, because we use linux PCs to implement the test system and a PC usually has limited number of NICs (Network Interface Cards), it will be scalable if we can use any number of PCs. For this purpose, a module named Distributed Communication Utilities (DCU) is added to support distribution of test components on several PCs. The PC on which MTC is located is called "main tester", other PCs are slave testers. PTCs can be distributed on the main tester and slave testers. All the testers sit on an ethernet LAN which is separated from networks connecting the testers with the IUT. We have implemented the DCU using TCP sockets API. Each PTC connects to MTC through a TCP connection, then receives commands and returns test results via this connection.

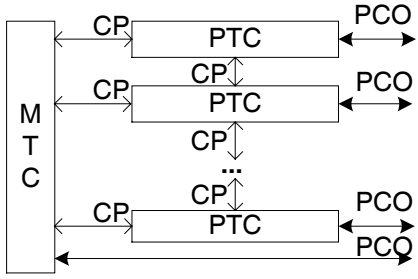

Fig. 4. TTCN-2 Test Architecture

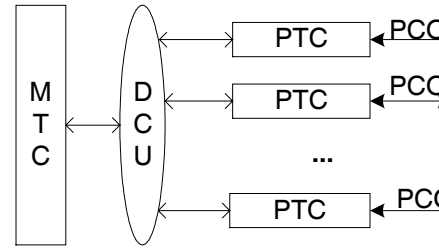

Fig. 5. The PADTACC Test Architecture

DCU is also used to access the CLI or SNMP module of the IUT to perform automatic configuration and monitoring functions during the test process, as explained later.

This specialized test architecture, named PADTACC – Parallel And Distributed Test Architecture with Centralized-Control, is shown in Fig.5. The simplification and patching is a reification of TTCN-2 test architecture when applied in routing protocol testing.

4 The RIPTS Test Notation

4.1 Requirements on Test Notation

Flexible and Simple.
In order to test the IUT we need to specify the test actions. Despite the wide applicability of TTCN, it is clumsy to manipulate the ordering randomness of incoming packets or pieces of routing information contained in a packet. Actually, for purpose of testing a specific protocol, a simple script language is enough. In addition, it should be easier to implement and flexible to be customized to meet the special test requirements of routing protocols.

Send, Receive and State Verification Facilities.
FSM-modeled protocols are tested using a transition-cover mechanism [11]. The test notation should provide capability to test a state transition: send to the IUT, receive from the IUT, verify its state.

Formalization and Automation of Test Configurations.
Test configuration includes configuration of the tester and the IUT. It is done before and in a test run because different test cases may use different test configurations. IP, TCP and many other protocols have only a small set of simple configuration tasks, many test cases share one test configuration which is recorded informally and done manually. Routing protocols such as OSPF, however, have many configuration scenarios. Furthermore, it is common that one test case uses several test configurations. This situation necessitates formalization and automation of test configuration operations. We choose to record the test configuration information in the test case and have it done automatically. The test notation should support this improvement. This approach greatly speeds the testing process.

Simulated Router.
In the past, we have overused the test specification language TTCN with a strong wish to keep the test suite pure. We almost walked off our legs until we realized that it is better to program complex protocol behaviors by C language. For example, OSPF neighboring routers have to experience a complex initial synchronization procedure before they enter the normal phase of exchanging further routing information. We implement Simulated Router encapsulating these complex behaviors in C language.

Simulated router (SR) can be viewed as a bipartite routing protocol implementation. The network side, which communicates to the IUT, is compliant with the protocol specification. In case there are state machines defined (in OSPF, BGP), an SR should implement the state machines. If no state machines are defined in the protocol (RIP), SR can be a simple encoder/decoder of the protocol messages. Additional functions such as authentication mechanism, checksum generation and verification, protocol configuration should also be implemented. However, SR need not possess all functions of a real router. For example, it neither originates routing information nor calculates the routing table. All the routing information to be sent to and received from the IUT is specified in the test script. As we will explain next, this is the way we simulate network topologies. The program side, which interfaces with the test script interpreter, acts as a PTC running associated test sequences under the control of MTC. Each network interface of an SR corresponds to one PCO.

Simulated Network Topology.
The nature of routing information processing is to track the dynamic network topologies in order to update the routing table accordingly. Should the tester simulate a network topology by running all containing routers simultaneously? This is unnecessary because the IUT can not see what happen beyond its neighboring routers and it learns the topology through routing information relayed by its neighboring routers. Thus, a tester need only simulate the neighboring routers which hide the simulated network topology behind. Fig. 6 illustrates this principle.

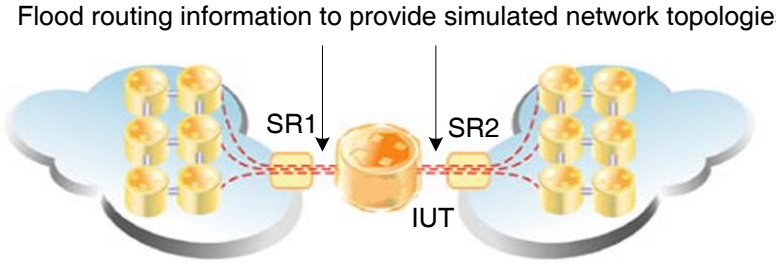

Fig. 6. SR hides the Simulated Network Topologies

4.2 The RIPTS Notation

This section informally presents the syntax, semantics of the RIPTS (Routing Information Processing Test Script) notation and formats of the test case file and test configuration file (see Fig.7). Note "{X}+" means "one or more X", "{X}*" means "zero or more X", "|" means "or", "%" introduces a parameter.

Definition 2 (test suite, test case, test step, test event) A set of test cases for a particular protocol composes a **test suite**. A **test case** is an implementation of a test purpose and comprises several test steps. A **test step** specifies a test configuration and a set of test events run under that configuration. A **test event** is the smallest indivisible unit of a test suite, such as transmission or reception of a packet.

To each test case run, we assign a verdict specifying whether the IUT is correct wrt this test purpose. PASS is assigned if the IUT passes all the test steps in the test case, FAIL otherwise. We say the IUT passes a test step if and only if all the test events in the test step are executed successfully according to their semantics (defined later).

In a test step, **TestCfg** statement designates the current test configuration file and **TestEvents** statement block designates the test events. By embedding test configuration information into test cases, the formerly stated requirement, formalization and automation of test configurations, is satisfied. Putting all test configuration information of a test step in an individual file enables reusability and modularity.

```
TestCase %case_index
{
 TestCfg %testcfg_file
 TestEvents
 {
    wait %pco_index %state |
    %pco_index send %ri_file |
    %pco_index recv %ri_file |
    %pco_index flush %ri_file |
    pause |
    stop {%pco_index}* |
    checktable %rib %table
 }+
}+
```

Format of a Test Case File

```
TestCfg %cfg_index
PCO %pco_number
IutCfg %iut_address %iutcfg_file
{
 Tester %tester_address
 {
  SR %srcfg_file %base_pco
 }+
}+
```

Format of the %testcfg_file

Fig. 7. Formats of some RIPTS Files

In a test configuration file, **PCO** statement gives the number of PCOs used by this test step, **IutCfg** statement specifies the IUT configuration where *iut_address* is an IP address of the IUT on the tester-LAN and *iutcfg_file* is the IUT's configuration file. Next statements are used to specify the configurations of testers and simulated routers. Remember that simulated routers act as PTCs and may be distributed on several testers. Each **Tester** statement block gathers all the SRs on one tester whose IP address on the tester-LAN is *tester_address* (used for startup of the SRs). Inside each **Tester** statement block are several **SR** statements which specify *srcfg_file* (the configuration of this SR) and *base_pco* (used to indexing the PCOs associated with this SR). PCO indexing is global. *iutcfg_file* and *srcfg_file* are protocol configuration files. The format of *srcfg_file* depends on SR since it is used by SR. *iutcfg_file* can use the format of *srcfg_file* and serve as the universal configuration script of the IUT. Users either configure the IUT manually according to *iutcfg_file* during the testing or design an automatic configuration mechanism such as using *expect* to access the *telnet virtual terminal* provided by the OAM module of the router under test.

Next, we look into the key construct: test event.

1. **wait %pco_index %state** Informs the relevant PTC (SR) to interact with the IUT until the possible Network Communication FSM reaches the named state. Previously we have written this procedure in TTCN test steps but found it very difficult and error-prone. So, this test event accomplishes the same function of the preamble test steps transferring the IUT to an appropriate starting state in TTCN.
2. **%pco_index send %ri_file** Send all the routing information listed in *ri_file* out of the named PCO to the IUT. It can be used to send many pieces of routing information at one time triggering sequential state transitions. **send** event finishes successfully when all the routing information is sent out.
3. **%pco_index recv %ri_file** Receive routing information from the IUT over the named PCO and see if it matches that specified in *ri_file*. We do not care the ordering of pieces of routing information so it's a multiple-packets and set matching. This test event finishes successfully if we receive all the given routing information within a limited time period, unsuccessfully otherwise. There is a special case when *ri_file* contains no routing information at all. This means no routing information is expected to arrive at this PCO.
4. **%pco_index flush %ri_file** Inform the relevant PTC to flush previously installed routing information via the named PCO. In RIP, the IUT will get a list of route entries with distance 16; in OSPF, the IUT will get a list of MAXAGE LSAs specified in *ri_file*; in BGP, the IUT will get a list of withdrew routes. **flush** is a special **send** event.
5. **pause** The only constructive element in the RIPTS notation, used to separate checking rounds (introduced later) and synchronize PTCs. Also used as a part of user interface implementation: pause the test execution and prompt to proceed. This is useful when we need to configure the IUT manually before go on testing.
6. **stop {%pco_index}*** Shut down the named PCOs. This includes flushing all the routing information previously sent out of named PCOs in *pco_index* list, and then disconnecting with the IUT. An empty *pco_index* list equals an all-PCO list. This test event is similar to the postamble test step returning the IUT to the initial state in TTCN.
7. **checktable %rib %table** *rib* and *table* specify the contents of the RIB and the routing table respectively. They should be verified at the end of each checking round. This state verification is comparable to the UIO verification in formal FSM testing.

Note that test events are not guarded by a timer explicitly. Relevant timers are put in a file together with other test options or parameters and read in a test run. Practice has shown that this approach simplifies writing test cases and enables easy adjustment of test options or parameters.

4.3 Structure of a Test Step

Definition 3 (a checking round) In testing, the tester initiates a sequence of input actions α which makes the IUT move from state s_i to state s_j while originating a sequence of output actions σ. This is called a checking round.

Apparently a checking round is a path of several transitions in the transition diagram of the MP-FSM given by Def. 1. A test step comprises several checking rounds separated by **pause** event. We use // to denote parallel relation, >> sequential relation, | alternative relation. Then the timing structure of a test step can be formalized as follows:

$B = b_1 >> b_2 >> .. >> b_n$, b_i (i=1..n) represents a checking round comprising some parallel test processes:

$b_i = a_{p1} // a_{p2} // .. // a_{pm}$, i=1..n, a test process a_{pk} (k=1..m) is a sequential execution of test events on PCO_k:

a_{pk} = **wait_sync** P_k S_l| **stop** P_k | (P_k **send** U_k >> P_k **recv** U'_k) | (P_k **flush** U_k >> P_k **recv** U'_k) | P_k **recv** U'_k

k=1..m, P_k denotes PCO_k, U_k is the routing information sent out of P_k, U'_k is the routing information received at P_k. S_l is a state of the possible Network Communication FSM.

Thus, a test step is interpreted as a sequential execution of several checking rounds which are parallel execution of several test processes according to above timing relations. In addition, use the **checktable** event at the end of each checking round to verify the end state if necessary.

5 IRIT – IP Routing Information Tester

We implemented all these ideas in a linux-based software tester – IRIT. Fig. 8 shows the software architecture which consists mainly of seven component modules: UI (User Interface), TM (Test Management), TE (Test Execution), DCU (Distributed Communication Utilities), SR (Simulated Router), Test Case Editor (TC Editor), Routing Information Generator (RI-Gen). AC is the abbreviation for Auxiliary Channel. Their roles and functions are presented briefly next.

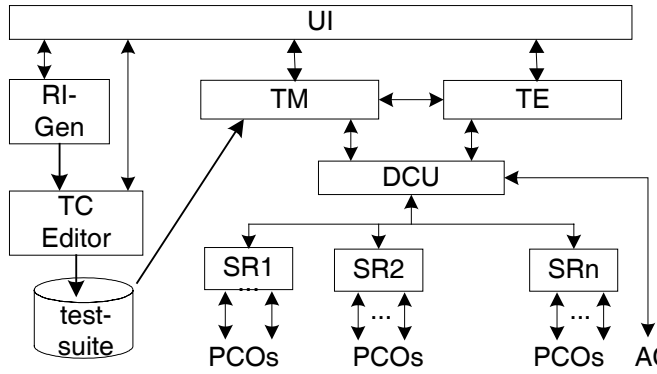

Fig. 8. the Architecture of IRIT

UI is a general designation of all the interfaces between modules and the user (the tester operator). RI-Gen supports manual routing information edit as well as auto-

matic routing information generation. Test Case Editor is used to edit and manage the test case files, test configuration files, protocol configuration files. TM plays an important role in the test organization. It implements the test selection interface by which the user may choose which test cases to run. Also, TM is responsible for the distribution of test configuration files on different testers.

The function of Test Execution is to interpret and execute the test events of a test case. It implements MTC in the PADTACC test architecture, runs the main test process and schedules the PTCs to run parallel test processes. A Simulated Router implements a PTC and its function has been detailed in sect. 4. DCU is used for the communications between test components and used to communicate with the IUT in other ways for other purposes, e.g. for remote configuration of the IUT.

6 Example Test Cases

Fig. 9 illustrates three test topologies. Each corresponds to an OSPFv2 test case. SRx denotes simulated router, Rx for virtual router (that "exists" in the simulated networks). ABR indicates area border router, ASBR for AS border router. Thick short lines labeled with N1, N2, … are multi-access networks. Lines connect interfaces to networks. OSPF areas are partitioned spatially and denoted by A0, A1 and so on.

TOP2 is used to test summary-LSA origination of the IUT under various area route aggregation and suppression configurations. This test case is typical in that it consists of five test steps with different test configurations. Fig. 10 shows the configurations of the IUT and what summary-LSAs should be originated by the IUT into area A1 and A0 (or A2). TOP3 is used to test that the IUT should originate ASBR-summary-LSA and ASE-LSA into normal areas but not into stub areas. The corresponding test case is shown in Fig.11 with comments. TOP4 is used to test the behaviors of the IUT with a virtual link configured. The test case is omitted here.

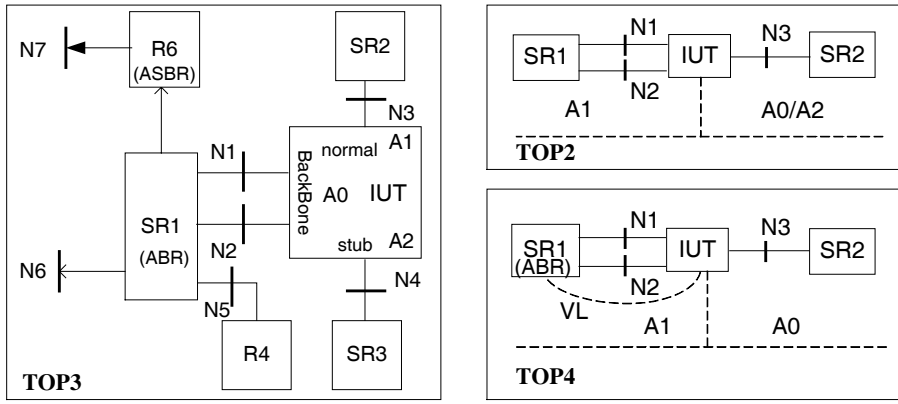

Fig. 9. Example Test Topologies of OSPFv2

N1: 10.1.1.0/24; N2: 10.1.2.0/24; N3: 10.1.3.0/24
IUT interface1: 10.1.1.5, metric=1; interface2: 10.1.2.5, metric=2;
interface3: 10.1.3.5, metric=3

Test Step	**IutCfg** (IUT configurations)	**summary-LSA** (route prefix, metric) the IUT should originate into this area
1	A0: no aggregate	(10.1.1.0/24, 1) (10.1.2.0/24, 2)
	A1: no aggregate	(10.1.3.0/24, 3)
2	A0: 10.0.0.0/12	(10.1.0.0/16, 2)
	A1: 10.1.0.0/16	(10.0.0.0/12, 3)
3	A0: 10.1.3.0/24 suppress	(10.1.0.0/16, 1)
	A1: 10.1.0.0/16; 10.1.2.0/24 suppress	-
4	A0: 10.1.0.0/16 suppress	(10.1.1.0/24, 1)
	A1: 10.1.0.0/16 suppress 10.1.1.0/24	-
5	A1: no aggregate	(10.1.3.0/24, 3)
	A2: stub, default-cost=10	(10.1.1.0/24,1) (10.1.2.0/24, 2) (0.0.0.0/0, 10)

Fig. 10. The TOP2 Test Case Outline

Test Case Statements	**Comments**
TestCase 3	pco1, pco2 belongs to SR1, pco3 belongs to SR2, pco4 belongs to SR3 as specified in *testcfg1*. A2 is a stub area configured not to import summary-LSAs.
TestCfg testcfg1	
TestEvents	
wait all full	wait completion of synchronization on all PCOs.
pco1 recv top3_a0_noaggr.ol	specify what LSAs are expected on each PCO, .ol files list the LSAs expected.
pco2 recv top3_a0_noaggr.ol	
pco3 recv top3_a1_noaggr.ol	
pco4 recv top3_a2_stub.ol	
pause	signal the end of a checking round
pco1 send top3_total5.ol	SR1 originates a rourter-LSA. It also simulates routes to R6, N6 and N7. The IUT should take N2 as the next hop network to R6, this is reflected in the LSAs originated into area A1.
pco1 recv empty.ol	
pco2 recv top3_total5.ol	
pco3 recv top3_total4.ol	
pco4 recv empty.ol	
stop	stop this test case

Fig. 11. The TOP3 Test Case

7 Summary

The main functions of a routing protocol can be divided into two parts: network communication (NC) and routing information processing (RI-Pro). The RI-Pro part is primary and should be the main content of routing protocol conformance test. This paper develops a pragmatic approach to test RI-Pro functions of IP routing protocols including RIP,BGP,OSPF [1].

An MP-FSM model is proposed to describe routing information processing of IP routing protocols. Some simplification and patching is made to TTCN-2 test architecture, resulting in the Parallel and Distributed Test Architecture with Centralized-Control (PADTACC). Compared to decentralized control, centralized control is simpler to realize the coordination and synchronization of test processes.

As a general test notation, TTCN has some limitations and extensions are not easy. We have designed a simple test notation called Routing Information Processing Test Script (RIPTS). RIPTS has advantages over a general test notation (e.g. TTCN) in the aspects of easy-understanding, extensibility, test efficiency and reliability, etc. Some limitations of TTCN-2 will be overcome in TTCN-3. For example, the newly introduced "external function" allows the use of functions that are written in other languages. It also provides a means to configure and monitor the IUT automatically in a test run.

All these ideas are implemented in a linux-based software tester - IP Routing Information Tester (IRIT). Compared with commercial testers like [8,9], IRIT is low-cost, easy to use. Also, it supports use of any PCs to compose a test system and thus gets rid of the limitation on available network interfaces. Formalized and automatic test configuration is another highlight that achieves faster test speed.

Future work based on IRIT exists in the following issues: stress testing, network topology simulation, routing information test generation. In addition, an effort will be taken to relate our work with TTCN-3 and translate the test cases to TTCN-3.

Acknowledgements

This research is supported by National Natural Science Foundation of China under Grant No. 90104002 and No.60102009. The authors would like to thank the anonymous reviewers for constructive comments on this paper.

References

1. RFC1058, RFC2328, RFC1771. URL = http://www.ietf.org/rfc.html
2. Wang Lirui, Ye Xinming. A Formal Approach to Conformance Testing of OSPF Routing Protocol. Proc. of the 6th Asia-Pacific Conf. on Communications, Page(s): 1286-1290
3. Jun Bi, Jianping Wu. A concurrent TTCN based approach to conformance testing of distributed routing protocol OSPFv2. Proc. of 7th Int. Conf. on Computer Communications and Networks. Page(s): 760 –767, 1998

4. Zhao Yixin, Wu Jianping, Yin Xia, et al. Test of BGP-4 based on the Protocol Integrated Test System. Proc. of 6th Asia-Pacific Conf. on Communications. Korea, 2000, 347-355
5. Wu Jianping, Zhao Yixin, Yin Xia. From Active to Passive: Progress in Testing of Internet Routing Protocols. KLUWER ACADEMIC PUBLISHERS. Proceeding of FORTE/PSTV 2001, Korea. 2001. 101-116
6. ISO/IEC 9646. Information Processing Systems, Open System Interconnection, OSI Conformance Testing Methodology and Framework- Part 3: The tree and tabular combined notation. 1998.
7. HAO R B, LEE D, RAKESH K. et al. Testing IP Routing Protocols - From Probabilistic Algorithms to a Software Tool. KLUWER ACADEMIC PUBLISHERS. Proc FORTE/PSTV2000. Pisa, Italy. 2000, 249-266
8. Agilent Technologies. RouterTester. http://www.agilent.com/comms/RouterTester, 2001
9. NetCom Systems. http://www.netcomsystems.com. 2000
10. Luo, G., Dssouli, R., Bochmann, G.V.etc. Test generation for the distributed test architecture. International Conference on Information Engineering '93, Page(s): 670 -674 vol.2
11. Lee, D., Yannakakis, M. Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE, Volume: 84 Issue: 8, Aug. 1996. Page(s): 1090 –1123
12. Expect Home Page. URL = http://expect.nist.gov