A Formal Approach to Conformance Testing of Distributed Routing Protocols

Jun Bi and Jianping Wu

Dept. of Computer Science, Tsinghua University, Beijing 100084, P. R. China

Abstract: This paper proposes a formal approach to conformance testing for OSPF v2, a widely used distributed routing protocol in Internet. Routing is performed in a distributed way, and, consequently, testing is more involved. For routing function testing concurrent TTCN handles concurrent test behaviors and is suitable for our needs. We first discuss a concurrent TTCN based OSPF test architecture. Then we present a specification model CEBE to specify an OSPF entity and to generate (combine the data and control flow) test suite. Finally, we design and implement a concurrent TTCN based test system to test OSPF implementations.

Key words: Conformance Testing, Concurrent TTCN, Routing Protocols

1. INTRODUCTION

Protocol conformance testing involves checking correctness of protocol implementations with respect to its (formal or informal) specification or requirements. Conformance testing is indispensable for the interoperability and performance of the protocol implementations. OSPF (Open Shortest Path First) is an important routing protocol for Internet. The network performance and reliability depend on the correct implementation of routing protocols of the routers. On the other hand, routing in the Internet is performed in a distributed fashion among different Autonomous Systems (AS) and in different areas (not in only a single router). How to formally specify and test concurrent behaviors of OSPF is a challenge to both

* This work has been supported by National Natural Foundation of China under Grant No. 69682002 and No. 69725003.

In this paper, we propose a formal approach to test OSPF protocols based on Concurrent Tree and Tabular Combined Notation (C-TTCN). A router running OSPF could be regarded as an entity in a distributed system [1]. An entity in a distributed system is associated with a set of interaction points and a set of concurrent behaviors at these interaction points, and it is more complex than the traditional peer-to-peer protocol entity [2][3]. To generate C-TTCN test suite, we present an entity specification model called Concurrent External Behavior Expression (CEBE). CEBE could directly specify concurrent external behaviors (both the I/O actions and data) at different interaction points of an entity in a distributed system and reduce internal states, actions and data. Therefore it is an ideal model to formally specify the entity in a complex distributed system and to generate (combine the data and control flow) C-TTCN test suite. The above methods have been implemented in an integrated test system PITS.

The rest of this paper is organized as follows. In section 2, we describe a C-TTCN based OSPF test architecture. Then we present the test suite generation method in section 3. Section 4 discusses the OSPF test suites and test system design. Section 5 concludes this paper.

2. TEST ARCHITECTURE

Figure 1. TC model of concurrent TTCN

OSPF v2, herein abbreviated simply as OSPF, is an IPv4 routing protocol documented in RFC 2328 [4]. It is a link-state routing protocol and all routers run the exact same algorithm in parallel. From the network topology database, each router constructs a tree of shortest paths with itself as the root. This tree defines a route to each destination in the AS. Externally derived routing information appears in the tree as leaves. Each individual piece of database is a particular router's local state (e.g., the usable interfaces and

reachable neighbors). The router distributes its local state throughout the AS by flooding. OSPF adjusts quickly to network topology changes, utilizing a minimum of routing protocol traffic.

TTCN is recommended by ISO to describe abstract test suite. C-TTCN is an extension of TTCN, and it allows a test system to execute test cases by several test components (TCs) running in parallel. As figure 1 shows, a tester consists of exactly one main test component (MTC) and any number of parallel test components (PTCs). TCs are linked by coordination points (CPs) to convey coordination messages (CMs). Communications between TCs with environment are through points of control and observation (PCOs). C-TTCN is suitable for specifying systems that consist of several parallel testers or lower testers, which is common in router testing.

Figure 2. Test architecture

Test architecture is a description of the environment in which an IUT is tested. For a router entity e , there are k concurrent behaviors happening at interaction points. The router entity e conforms to the specification if and only if all these behaviors conform. Therefore, OSPF test architecture must deal with these concurrent behaviors at several interaction points.

We use the test architecture shown in figure 2. It consists of a test system, an entity implementation under test (IUTE), a test context, several PCOs and several interaction points. Test system is an implementation of a test suite based on C-TTCN. It carries out experiments by executing test cases and observing test results. Interactions between IUT and its environment take place at interaction points (IPs). Test system communicates with IUT indirectly: through *test context*. The points where tester communicates with test context, and indirectly controls and observes IUT, are called Points of Control and Observation (PCOs). Execution of a test case starts with the execution of MTC. It is the concern of MTC to set up all PTCs, and to

manage all PCOs and CPs to be connected to. PTCs can be created by MTC on demand. A ‘*create*’ operation associates a PTC with a behavior tree. The newly created PTC starts execution of its assigned behavior tree concurrently with MTC. MTC may explicitly terminate a PTC by executing a ‘*down*’ operation.

3. C-TTCN TEST GENERATION

3.1 Basics of CEBE

In this section, we describe an entity specification model CEBE. CEBE aims to improve specification abilities of concurrent behaviors (actions) and interaction points (gates). It is the extension of EBE presented in [5]. The essence of both EBE and CEBE is to describe data in addition to control behavior from the point of view of an external system. Then CEBE can directly specify concurrent external behaviors (both the I/O actions and data) at different interaction points of an entity in a distributed system and reduce internal states, actions and data. Therefore, it is an idea model to formally specify the complex object and to generate (combine data flow and control flow) C-TTCN test suite. A specification of CEBE can be obtained from ETS (the extension of ETS presented in [6]). [6] gives the translation from standard specification language LOTOS and Estelle to ETS. In [7], an ETS semantics for Z is presented. CEBE can be obtained from ETS by eliminating the internal actions. Therefore, the CEBE may be translated from the standard specification model. There are two phases in our test generation method: phase 1: Derive test sequence from CEBE; phase 2: Generate test suite in C-TTCN.

Definition 3.1 A *CEBE* is a 7-tuple $\langle S, G, \Sigma, A, R, s_0, v_0 \rangle$ where (1) S is a finite set of *external* states whose elements are pause states of interactions exchanged between entity and its external environment; (2) G is a set of gates over which a CEBE can communicate; (3) Σ is set of data on CEBE (see below); (4) A is a set of *external* actions on CEBE (see below); (5) R is a set of logic relations of transitions between external states (see below); (6) $s_0 \in S$ is the initial state of the entity, which represents initial external state; and (7) v_0 is a set of initial assignments (initial values of all variables and timers) of Σ .

Definition 3.2 $\Sigma = IO \cup VAR \cup CONS \cup TIMER \cup PARA$, is a set of data on CEBE, where (1) IO is a set of input or output events (i.e., the ASPs

and PDUs); (2) *VAR* is a set of variables of various types; (3) *CONS* is a set of constants of various types; (4) *TIMER* is a set of timers and (5) *PARA* is a set of parameters of events in IO. For an event $e \in IO$, the parameters $p_1, p_2, \dots, p_n$ are denoted as $e.p_1, e.p_2, \dots, e.p_n$, e is denoted as $e(e.p_1, e.p_2, \dots, e.p_n)$.

Definition 3.3 A behavior b is the elements of G with a list of input and output events and their value declaration. Then set of b is denoted as: $B = \{ \langle g_i, ? r_i (r_i.p_1, r_i.p_2, \dots, r_i.p_n), ! s_i (s_i.p_1, s_i.p_2, \dots, s_i.p_n) \rangle \mid g_i \in G, r_i, s_i \in IO, r_i.p_i, s_i.p_i \in PARA \}$, where symbol "!" indicates the output of an event s_i to external environment and symbol "?" indicates the input of an event r_i from the external environment. The absence of r_i or s_i could be denoted as symbol "--". For example $b = \langle g_i, ? r_i (r_i.p_1, r_i.p_2, \dots, r_i.p_n), ! -- \rangle$. Here we give some notations: (1) $b_1 >> b_2 >> \dots >> b_n$, the behaviors are sequential; (2) $b_1 [] b_2 [] \dots [] b_n$, the behaviors are equally valid alternatives (choice); (3) $b_1 || b_2 || \dots || b_n$, the behaviors are parallel.

Definition 3.4 An action a is set of concurrent behaviors. $A = \{ a_i \mid a_i \in A \}$, $a_i =_{\text{def}} b_j \in B \mid a_i [] a_i \mid a_i >> a_i \mid a_i || a_i \mid (a_i)$, where the operator precedence is ' $()$ ' > ' $[]$ ' > ' $>>$ ' > ' $>>>$ '.

Definition 3.5 Transition relation $R \subseteq S \times A \times S \times P \times \text{PowerSet}(O) \times \text{PowerSet}(F)$ is the set of transition relations where: (1) A is a set of actions; (2) S is a set of states of CEBE; (3) $P(\Sigma): \text{PowerSet}(\Sigma) \rightarrow \{ \text{TRUE}, \text{FALSE} \}$ is a set of predicates functions ($=, \geq, \leq, >, <, \wedge, \vee, \neg$, Timeout, et.al.); (4) $O(\Sigma): \text{PowerSet}(\Sigma) \rightarrow \text{PowerSet}(\Sigma)$ is a set of operation functions ($:=, +, -, *, /$, StartTimer, CancelTimer, et. al.); (5) $F(\Sigma): \text{PowerSet}(\Sigma) \rightarrow \text{PowerSet}(\Sigma)$ is a set of functions (i.e. max(), min(), et.al.).

The intuitive meaning of a transition $r \in R$ is that if a CEBE is in state s and the enabling action a is offered, then the enabling predicate p is evaluated on current assignment of variables. When p is true, the CEBE will go into a new state s' and the environment are updated by an operation o and a function f . The absence of predicate could be denoted as *TRUE*, the absence of operation or function is denoted as *NIL*.

3.2 Test sequence derivation

CEBE based test sequence derivation strategy is defined as two steps. First identifying all possible valid *IPs* (Interaction Path) from CEBE specification by imposing constraints set on *IPs*. Then generating *SIPs* (I/O

Subpath) from the above IPs through checking dependencies between transitions in each IP.

Step 1: An interaction path IP is the externally observable track on which a sequence of interactions between entity and its external environment occurs, starting from initial external state s_0 of the entity and ending in s_0 . Any loop (or cycle) interaction in an IP is traveled only once. IP with loop is an interaction path in which there exists one or more loop interactions.

Step 2: An I/O subpath SIP is the externally observable track $r_1, \dots, r_k$ in an IP , where:

1. $r_1 = \langle s_{r1}, a_{r1}, s'_{r1}, p_{r1}, O_{r1}, F_{r1} \rangle$ is the first transition relation when satisfies:

- a) a predicate $p_{r1}(\Sigma_{pr1})$, and/or
- b) a set of operation $\{ \dots, o_{r1}(\Sigma_{or1}), \dots \}$, and /or
- c) a set of function $\{ \dots, f_{r1}(\Sigma_{fr1}), \dots \}$;

2. $r_k = \langle s_{rk}, a_{rk}, s'_{rk}, p_{rk}, O_{rk}, F_{rk} \rangle$ is obtained when logical relations satisfy:

- a) $(\forall i, (\Sigma_{prk} \cap o_{rli}(\Sigma_{orli})) \neq \emptyset) \vee (\forall i, (\Sigma_{prk} \cap f_{rli}(\Sigma_{frli})) \neq \emptyset) \vee (\forall i, \forall j, (\Sigma_{orkj} \cap o_{rli}(\Sigma_{orli})) \neq \emptyset) \vee (\forall i, \forall j, (\Sigma_{orkj} \cap f_{rli}(\Sigma_{frli})) \neq \emptyset) \vee (\forall i, \forall j, (\Sigma_{frkj} \cap o_{rli}(\Sigma_{orli})) \neq \emptyset) \vee (\forall i, \forall j, (\Sigma_{frkj} \cap f_{rli}(\Sigma_{frli})) \neq \emptyset)$, and
- b) a predicate $p_k(\Sigma_{prk})$, and/or
- c) a set of operation $\{ \dots, o_{kj}(\Sigma_{orkj}), \dots \}$ and /or
- d) a set of function $\{ \dots, f_{kj}(\Sigma_{frkj}), \dots \}$;

3.3 C-TTCN test suite generation

Based on SIP, a test tree with nondeterminism, correctness, and defense branches can be derived, and three kinds of verdicts (PASS, FAIL and INCONC) are assigned to each leaf of the test tree. The test tree can be mapped to a test case in C-TTCN. Finally, these test cases are grouped according to some rules, and are mapped to a complete test suite in C-TTCN.

As discussed above, in the test case specified by C-TTCN, the behaviors of each test component are expressed with a *behavior tree (BT)* in a test case where BT is a tree-like presentation of the temporal relations between test events. In order to get formal definition of the semantics, we need to introduce the Test Behavior Expression (TBE) to express the behaviors of test components in algebraic form.

Definition 3.6 The syntax of a TBE

$TBE \ B =_{def} stop \mid exit \mid id?a;B \mid id!y;B \mid B \parallel B \mid B >> B \mid [q];B \mid (I:=val);B \mid (B) \mid start \ tid \ val;B \mid cancel \ tid;B \mid timeout \ tid;B \mid create \ id \ B \mid done \ id$

The operator precedence is ' $;$ ' > ' $[]$ ' > ' $>>$ '. ' $stop$ ' denotes termination of a running, ' $exit$ ' denotes a successful quitting from an attached TBE. ' $id?a;B$ ' and ' $id!y;B$ ' represent respectively a receiving and a sending event in PCO or CP identified by ' id '. ' $;$ ' denotes sequential composition. ' $[]$ ' denotes choice. ' q ' is the qualifier which is a boolean expression, its value is either *TRUE* or *FALSE*. ' $I:=val$ ' is an assignment, where ' I ' is an identifier and ' val ' is a value. ' $>>$ ' denotes attaching operation. ' $start$ ' and ' $cancel$ ' is a pair of timer operator, ' $timeout$ ' represents the expiration of a timer. ' $create\ id\ B$ ' creates a PTC named id whose behavior tree is B , while ' $done\ id$ ' waits until the PTC named id has finished its running.

To understand the algorithm we use TBE to describe the rules for C-TTCN generation. In the real tool TUGEN, the test suite in C-TTCN MP is generated.

Rule 1: Each SIP is mapped to one C-TTCN test case;

Rule 2: In one SIP, if there are several relations $r_1, r_2, \dots, r_n$, then attach the tree $i+1$ to tree i .

$TBE_M \square r1 >> r2 >> \dots >> r_n$

Rule 3: In one action a of relation r , if $b_1 >> b_2 >> \dots >> b_n$, then $b_1, b_2, \dots, b_n$ are behaviors and it is successfully completed if b_{i+1} happens immediately after the successful completion of b_i ($i=1, 2, \dots, n-1$):

$TBE_M \square create\ P1\ TBE_1; start\ t1; (CP1?CM_1; down\ P1; stop)\ []\ (timeout\ t1; stop)$

$TBE_1 \square b1 >> b2 >> \dots >> b_n >> (CPi!CM_i; stop)$

Rule 4: In one action a of relation r , if $b_1 [] b_2 [] \dots [] b_n$, then an alternative is matched if the behavior b_i corresponding to the alternative happens first:

$TBE_M \square create\ P1\ TBE_1; start\ t1; (CP1?CM_1; down\ P1; stop)\ []\ (timeout\ t1; stop)$

$TBE_1 \square (b1 >> (CPi!CM_i; stop))\ []\ (b2 >> (CPi!CM_i; stop))\ [] \dots []\ (b_n >> (CPi!CM_i; stop))$

Rule 5: In one action a of relation r , if $b_1 || b_2 || \dots || b_n$, then $b_1, b_2, \dots, b_n$ are started in parallel and the parallel tree completes only if $b_1, b_2, \dots, b_n$ complete:

$TBE_M \square create\ P1\ TBE_1; \dots; create\ Pn\ TBE_n; start\ t1; (CP1?CM_1; down\ P1; start\ t2; (\dots; (CPn?CM_n; donePn; stop) [] (timeout\ tn; stop)); \dots) [] (timeout\ t2; stop) [] \dots [] (CPn?CM_n; down\ Pn; start\ t2; (\dots) [] (timeout\ t2; stop)) [] (timeout\ t1; stop)$

$TBE_1 \square bi >> (CPi!CM_i; stop)$

Rule 6: In each behavior b of action a in a relation r , $\langle g_i, ? r_i (r_i.p_1, r_i.p_2, \dots, r_i.p_n), ! s_i (s_i.p_1, s_i.p_2, \dots, s_i.p_n) \rangle$, then
 $TBE_i : g_i ! r_i ; g_i ? s_i ; CP1 ! CM_1 ; stop$

Rule 7: In the declaration part, the *timer*, *variable* and *constant* could be mapped from *TIMER*, *VAR*, and *CONS* with the initial value from v_0 ;

Rule 8: In the declaration part, the *ASP*, *PDU* can be mapped from *IO* with the parameters *PARA* for the events;

Rule 9: In the constrain part, the *timer*, *variable*, *ASP* and *PDU* can be mapped from each event $e_i (e_i.p_1, e_i.p_2, \dots, e_i.p_n)$ in b with the value of parameters associated with the rules defined in section 3.1.

4. OSPF TESTING

4.1 Test Suite Design

We implement the above test method by an automatic tool TUGEN. The generated OSPF test suite has over 4000 SIPs when search depth is 6. We delete the redundant or useless ones and select 543 test cases that have clear test purposes. Table 1 shows the structure of OSPF test suite.

OSPFv2 Conformance Test Suite	
Test Group	Test Cases
DOWN	23
INIT	70
EXSTART	107
EXCHANGE	123
LOADING	68
FULL	152

Table 1 OSPFv2 Test Suite

In CEBE specification, we delete some internal states defined in the protocol, such as Attempt, 2-Way, and some interface state (Loop, Unloop) because these states are not observable externally. Moreover, we reduce some internal events defined in the protocol such as: Interface up/down and loop/unloop. The following events are replaced by the external input/output

PDU's (with different parameters) and timer events: Start, 2-WayReceived, NegotiationDone, ExchangeDone, BadLSReq, Loading Done, AdjOK?, SeqNumMismatch, 1-Way, KillNbr, InactivityTimer, and LLDn. We give examples of the C-TTCN based OSPF test suite. Here are two transition relations $r_{44(1)}$ and $r_{44(2)}$:

```

S4 [ <PCOI, ?--, !DDP0 (Ver,Type,Len, D:=MyID, AID, CSM, AT, Auth,
MTU, Option,I,M,MS, Seq:= YouSeq)> |
  PREDICATE: TRUE;
  OPERATION: HaveSendDDP:=1;
  START RtxDDPTM;
  FUNCTION: NIL ]*S4          {r4-4(1)}
+ [<PCOI, ?LSU(...), !LSAck(...)> >> (<PCO2, ?--, !LSU(...)> ||
<PCO3, ?--, !LSU(...)> || ... || <PCOn, ?--, !LSU(...)>) |
  PREDICATE: HaveSendDDP=1;
  OPERATION: NIL;
  FUNCTION: NIL]*S4          {r4-4(2)}

```

In $r_{44(1)}$, predicate is TRUE. In $r_{44(2)}$, predicate indicates that it needs "HaveSendDDP=1". It means that in state 4 (FLOOD), IUT must send last DDP first. The $r_{44(1)}$ is an associated transition. When it fires, associated function $f_{44(1)}$ and operation $o_{44(1)}$ is executed. $o_{44(1)}$ is to set HaveSendDDP=1 and start retransmit timer RtxDDPTM. When OSPF-CEBE variables and timers are set, DDP (with parameters ID and Seq are assigned) are assembled (their parameters are set) and sent. Then $p_{44(2)}$ becomes TRUE and it can be executed. Therefore, when generating test case, $r_{44(1)}$ and $r_{44(2)}$ should be in one SIP (test case). In $r_{44(2)}$, if IUT receives an Update packet, it will send an Ack packet, then it updates its database and floods this new information in an Update packet to all the neighboring routers. This flooding process is concurrent and is followed with the receiving of Update (we use >> and ||). The generated C-TTCN test case is shown as follows.

```

TREE_M: behavior specification of M
  CREATE(P1, TREE_P1)
  START T1
  CPI?CM_1
  ?DONE(P1)
  CREAT(P1, TREE_P1');
  START T2;
  CPI?CM_1
  ?DONE(P1')
  CREAT(P2, TREE_P2)

```


PITS. It provides an integrated platform to test for both protocols (OSI and TCP/IP) and entities (in distributed systems), based on C-TTCN (and its extension). This system conducts its testing process on the basis of TTCN/C-TTCN test suites, and parallel processing is explored to increase its efficiency. After test selection on the basis of PICS and PIXIT by TM (test management), test cases are interpreted and then executed by TE (test execution) step by step. TE is an engine that interacts with other components of PITS, and controls test process according to the content of TTCN or C-TTCN, and simultaneously generates all information required to produce test reports. Since SUT OSPF is an entity in the distributed routing area, there may be many PCOs and conceptual PTCs for these PCOs. By adjusting the reference implementation and test path, we can get flexible test configuration to meet the different need of entities in the distributed systems.

Figure 4. Architecture of test execution

TE can be regarded as an abstract TTCN/C-TTCN machine. It interprets and executes test cases strictly according to operational semantics of TTCN/C-TTCN. As shown in figure 4, TE is composed of an interpreter, one MTC, and zero or more PTCs. The interpreter gets the offset of a test case from TM, and then obtains an abstract test case from the test suite. The interpreter transfers an abstract test case to an executable test case by 3 steps. (1) *Syntax analysis*: check the syntax of an C-TTCN based test case; (2) *Static semantics analysis*: check the static semantics of the test case; and (3) *Transformation to an evaluation tree*: construct an evaluation tree by appending default behaviors, converting REPEAT constructs into GOTO constructs, and expanding attached trees.

MTC is the control part of test execution. It gets an executable test case from the interpreter and executes the test case by traversing a path in the evaluation tree of the test case. When meeting 'create' event, MTC creates a

PTC and assigns the behavior tree defined by 'create' event to PTC. Configuration declaration table determines the location of PTC. Initially, PTCs are demon processes listening to some special ports. After MTC executes 'create' test event, the corresponding PTC will be active and run test behavior tree conveyed from MTC. Each PTC keeps its own test log during executing of a test case. When the execution of the test case completes, it reports log to MTC. Then MTC reports all logs to test result module. The test execution is performed by the method presented in [8].

Figure 5 shows the OSPF test architecture with PITS when IUT is the OSPF router entity. The RT is optional and depends on the interface of router under test. This test architecture has been used to test router products such as CISCO 4700. CISCO passed most test cases. However, we found some problems such as DDP interactions and MTU setting with CISCO version 11.

Figure 5 OSPF router entity test architecture with PITS

5. CONCLUSIONS

In this paper, we have applied C-TTCN for OSPF routing protocol testing. The real testing is accomplished by using a C-TTCN based test system PITS to test CISCO 4700 router. C-TTCN OSPF test suite is derived by a tool TUGEN based on CEBE. From our experience, when we write OSPF CEBE specifications, we find many internal states and actions that can be reduced, simplified, or replaced by external events in CEBE. Currently, we are involved in the work of testing the whole OSPF AS and Area as a black box by observing its external behaviors to stimuli provided by simulations of

other Areas or ASs. Then it is necessary to specify CEBE of an entity Area or AS and to generate test suite in C-TTCN.

CEBE and C-TTCN based concepts and methods presented in this paper can also be used in the area of conformance testing in distributed systems. For instance, CEBE could be used to model communicating systems and entities of OSI, TCP/IP and ODP. Furthermore, we are studying CEBE model and C-TTCN generation algorithm to improve the test execution efficiency of the generated test suite. For instance, we want to combine TBEs of different PTCs in one test case to deduce the size of behavior tree.

REFERENCES

- [1] A.Ulrich, L.Logrippo and J. Meer, "An approach to testing distributed software systems", *15th Int. Conference on Protocol Specification, Testing and Verification*, IFIP, 1995.
- [2] A.A.F. Loureiro and S.T. Chanson, "Checking unstable properties in distributed testing of communication protocols", *10th Int. Workshop on Testing of Communicating Systems*, IFIP, 1996.
- [3] A.C.Y. Wong, S.T.Chanson, S.C. Cheung, and H.Fuchs, "An framework for distributed object-oriented testing", *10th Int. Conference on Formal Description Techniques for Distributed Systems*, IFIP, 1997.
- [4] J. Moy, OSPF version 2, RFC 2328, 1998.
- [5] J. Wu and S.T.Chanson, "Testing sequence derivation based on external behavior expression", *2nd Int. Workshop on Protocol Test Systems*, IFIP, 1989.
- [6] J. Wu and S.T.Chanson, "Translation from LOTOS and Estelle specifications to extended transition system and its verification", *2nd Int. Conference on Formal Description Techniques for Distributed Systems*, IFIP, 1989.
- [7] J. Derrick, E. Boiten, H. Bowman and M. Steen, "Supporting ODP - translating LOTOS to Z", *1st Int. Workshop on Formal Methods for Open Object-based Distributed Systems*, IFIP, 1996.
- [8] R. Hao and J. Wu, "A formal approach to protocol interoperability testing", *6th Singapore International Conference on Networks*, IEEE, 1997.