Interoperability Testing of VoIP Systems

Nancy Griffeth, Ruibing Hao, David Lee, Rakesh K. Sinha
Bell Laboratories, Murray Hill, New Jersey 07974
E-mails:{nancyg,rhao,lee,rks1}@research.bell-labs.com

Abstract— **Voice over IP (VoIP) systems have to be integrated and interoperate with the existing Telephone Network before they become widely adopted. However, different standards have been proposed and the prevailing standard (H.323) is incomplete, complex, and provides the implementers with "vendors latitudes". As a result, there is no guarantee that even H.323-compliant systems would interoperate properly. Interoperability testing has become indispensable. We present techniques for automatic generation of a minimal set of test cases, covering all the required system interoperations. Our algorithms have been implemented and incorporated in a software tool - ITIS. We report experimental results on VoIP systems interoperability testing, using ITIS.**

I. Introduction

With the rapid growth of Internet, more and more new protocols are proposed and new products are developed, such as Voice over IP(VoIP) and Virtual Private Network(VPN). When two or more entities in separate communicating systems are integrated and need to interact with each other to perform a certain task the capability to operate as desired is called interoperability, which is considered an essential aspect of correctness of integrated systems. However, products from different vendors or even from the same vendor often do not interoperate properly. Two main causes of non-interoperation are ambiguity of protocol specification and vendor's proprietary extensions. Interoperability testing is to check the interoperations among integrated system implementations.

Research works on interoperability testing can be roughly classified into two categories: general concepts and experiences of interoperability testing [4], [2], [13], and systematic generation of interoperability test suites [5], [12], [7], [11], [10], [3], [6]. Our work has been motivated by the need to test interoperability of systems carrying voice calls over the IP network. VoIP systems are to be integrated and interoperate with the existing Public Switched Telephone Network (PSTN) before they are widely adopted. We model the system behaviors by extended finite state machines (EFSM). EFSMs are finite state machines, extended with variables, and are compact representation of finite state machines [9]. We propose a testing architecture for interoperability testing. Based on the experiences from domain experts, we discuss various criteria that cover the required patterns of interoperations. A key idea in developing the coverage criteria is that interoperability errors are introduced only when the integrated systems are "interacting" with each other. Based on this observation, our test generation only requires information of part of the integrated communication systems; it is impossible to obtain, model and analyze the whole systems in practice. For instance, for VoIP interoperability testing, we only need information of end users required behaviors and H.323 interfaces. Consequently, it does not require the modeling and construction of the composition of all the integrated systems, including the IP network, and avoids the state explosion problem while checking all the required interoperations.

We design efficient data structures and test generation algorithms, which guarantee desired coverages and contain a minimal number of tests. These algorithms are part of a software tool, ITIS, developed at Bell Laboratories. With this tool, we have generated interoperability test cases for: End users vs the rest of the communication system; and End users and H.323 vs the rest of the communication system. For clarity, we only report the experimental results of interoperability testing of the end users behaviors vs the rest of the systems.

The remaining of this paper is organized as follows. After a general discussion of interoperability testing in Section 2, we present interoperability test generation algorithms along with a range of coverage and redundancy criteria in Section 3. Section 4 contains a description of ITIS, a software tool based on the algorithms developed, and reports experimental results of applying ITIS to VoIP systems.

II. Interoperability Testing

When we conduct interoperability testing, we are concerned only with those failures that occur when integrated systems are interoperating. For instance, consider Voice over IP systems in Fig. 1. Two H.323 end-users want to communicate. They each access the communication system through a H.323 gateway, using H.323 protocol. Consider the case that we want to test the interoperability between H.323 end users and the rest of the systems, which is the H.323 gateways and the communication system.

Fig. 1. Model of interoperating systems

For this interoperability testing, we would like to test all possible system interoperations between the end users vs the rest of the system, assuming that we can only access the system through end user behaviors. We model the joint behaviors of the two end users by an EFSM. Ideally, we want to check every input sequence to the end user EFSM so that all its interoperations with the rest of the system are tested. However, this is an impossible task; there are infinitely many such sequences. Instead, we remove the redundant tests (redundancy will be defined in the sequel) and reduce to a small number of tests.

0-7803-6451-1/00/$10.00 © 2000 IEEE

In general, we want to test interoperations of system A and B. We have complete information of A but not B or choose not to model B, either because the information is not available or it is too complex. We can only access the systems through the interface with A. Based on our information about A, we want to construct a set of irredundant tests such that all the interoperations between A and B are tested. Our goals are: (1) Completeness: all the interoperations between the two systems are tested; and (2) Irredundancy: all the redundant tests are removed to minimize the total number of tests.

Note that interoperability testing is different than conformance testing; even a checking sequence [9] of system A may not be sufficient due to the following reasons. A checking sequence guarantees structural isomorphism between the implementation and specification machines under the assumption that the implementation machine has no more states than that of the specification machine. It may not be true for the expected behavior of an integrated system, in which the number of states is unknown. Furthermore, we have no information or only partial information of system B. Therefore, for interoperability testing, we need different tests to ensure the coverage of all the interoperations between A and B, given only information of A.

There are various techniques to remove redundancy. For instance, in the configuration of Fig. 1, a key idea in developing the test cases is that interoperability errors will be introduced only when the gateways are actually talking to each other about a call. Thus we can ignore local activities involved in the protocol, such as obtaining information from an end-user. This is illustrated in the call flow for a simple telephone call in Fig. 2. The transition Off-hook from Idle state to Dialing state is local, because it involves only the originating gateway and the end user. The Dial transition from Dialing to Calling involves interoperations, because the state is changed in both gateways. Similarly, the Off-hook transition which connects the call changes the state in both gateways from Calling to Connected. We generate test cases, which are focused on transitions involving interoperations.

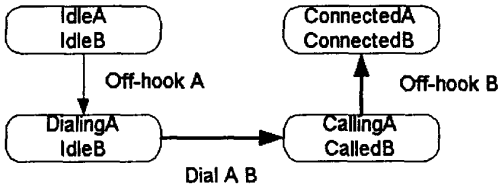

Fig. 2. Partial call flow of a telephone call

III. Redundancy Criteria and Test Generation

We want to test interoperability of system A and B, assuming we have complete information about A. We can model it by an EFSM and construct its reachability graph, which is a directed graph, a transition diagram [9]. Throughout the rest of the paper, we will be using graph terminology. For example, we will denote the initial state as the "source node", transitions as "edges" etc.

Ideally, we want to cover (test) all possible execution sequences, often called scenarios. Therefore, covering all possible execution sequences requires that we cover all the branches and all the possible paths. Consequently, we cover all the interoperations from A to B, and this would be our interoperability coverage criteria. However, it is impossible; in general, transition diagrams often contain cycles and, therefore, will have infinitely many distinct paths. We need to remove the redundant paths, if possible, and reduce the number of tests to a manageable size.

Our test generation process contains three steps:

step 1 Generate all possible acyclic paths, i.e. paths without any repeated vertices.

step 2 Generate all possible simple cycles, i.e. cycles that do not contain any smaller cycles.

step 3 "Combine" the paths (from step 1) and cycles (from step 2) to generate the final set of paths.

The number of paths and cycles generated in the first two steps is finite. The goal of the "combine" process in step 3 is to generate a finite number of paths that cover all necessary scenarios. This is done as follows:

First we include all the acyclic paths. Then we generate additional cyclic paths as follows. For each acyclic path P, find all the cycles that share a node with P. Let these cycles be $C_1, C_2, \ldots, C_k$ and let v_i , $1 \leq i \leq k$, be a node common to C_i and P. Then generate a new (cyclic) path by replacing node v_i in P with cycle C_i.

The algorithm for steps 1 and 2 is called EXHAUSTIVE-COVERAGE and described in the full version of this paper[8]. This algorithm has a transition graph G of an FSM as input; it finds all the Strongly Connected Components(SCCs) in the graph and shrinks the graph into a Directed Acyclic Graph(DAG); the output from this algorithm is a minimal set of acyclic paths providing exhaustive coverage of the graph G.

A. Redundancy Criteria

There is a finite number of distinct paths (tests) in a DAG. However, we do not want to test all of them; Our goal is to generate a minimal number of tests, which satisfies the coverage criterion and does not contain any *redundant* tests.

As stated in the introduction, a key insight is that interoperability errors will be introduced only when the gateways are actually talking to each other about a call. We label the transitions as either "white" (local activity) or "black" (involves both gateways). Thus for interoperability test generation, we do not need to cover white transitions. In addition, a very popular *redundancy* criterion in practice is: to consider only paths that start from the source node and are not proper prefix of another test. This motivates the following redundancy criteria.

(R1) Proper prefixes are redundant and can be discarded.

(R2) Remove all-white test sequences.

(R3) Let (u, v) be the first black edge in the test sequence. Then replace the path from the source node to node u with the shortest path between the source node and node u.

(R4) Let (u, v) be the last black edge in the test sequence. Then replace the path from node v to the sink node with the shortest path between node v and the sink node.
(R5) If there is a sequence of white edges, in which each party separately reaches the idle state, terminate the sequence at the last black edge and then use a shortest path to the state "IdleA,IdleB" at the end of the sequence.

The first three criteria reflect our belief that white transitions are not relevant for interoperability testing. Their only use is to connect (relevant) black transitions. For example, $R3$ states that the sequence of transitions before the first black transition is not relevant to us, so we might as well replace that part with the shortest all-white sequence. Redundancy criterion $R4$ is dual of $R3$. The last criterion recognizes that if both states reach the "idle" state through a sequence of non-relevant transitions then whatever happens in the rest of the sequence will be tested at the beginning of another generated sequence.

The Algorithm ADEQUATE-COVERAGE in Fig. 3 generates all acyclic paths satisfying these criteria.

A.1 Generating Test Sequences according to $R1$-$R5$

The general idea of the algorithm is to consider all possible black transitions that can be the first (resp. last) in any sequence and add a marker 'super-edge' for the shortest all-white path from the source node (resp. to the sink node).

ADEQUATE-COVERAGE

Input. transition graph G of a FSM with a designated source state.
Output. complete set of paths according to redundancy criteria $R1$-$R5$.

```
for each node v that has an outgoing black edge
    if there is an all-white path from the source node to v then
        add a super-edge from the source node to v;
delete all outgoing edges (not super-edges) from the source node;
for each node v that has an incoming black edge
    if there is an all-white path from v to a sink node then
        add a super-edge from v to this sink node;
delete all incoming edges (not super-edges) to the sink node;
generate all acyclic paths in the resulting graph with the
modification that no super-edge should follow or precede a
white edge;
replace each super-edge with the shortest all-white path in the
original graph; /* this may introduce some loops */
process each path to get rid of loops;
process each path according to redundancy criterion R5;
```

Fig. 3. Generating paths for adequate coverage (according to $R1$-$R5$)

Fig. 4 shows how Algorithm ADEQUATE-COVERAGE is executed on an example FSM.

B. Additional Redundancy Criteria

On most practical systems, we expect Algorithm ADEQUATE-COVERAGE (Fig. 3) to generate a considerably smaller set of test sequences than the naive algorithm. Unfortunately even such a smaller set may be too large for manual test execution.

In this section, we discuss a set of more restrictive criteria for test generation that concentrates only on black transitions.

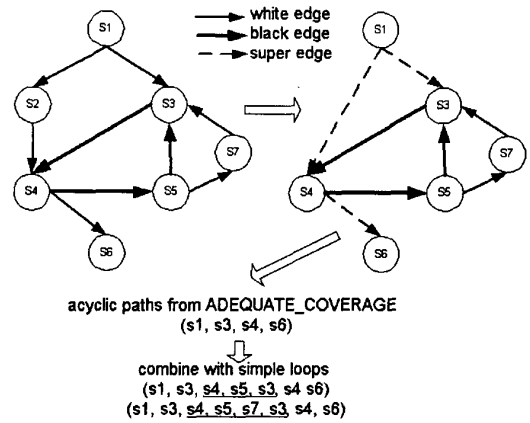

Fig. 4. Example test generation using Algorithm ADEQUATE-COVERAGE

(R6) Generate acyclic paths that consist only of black edges, except that the prefix from the source node and the suffix to the sink node is allowed to contain white edges.
(R7) Generate simple cycles consisting only of black edges.

B.1 Generating Test Sequences according to $R6$-$R7$

We now present Algorithm BASIC-COVERAGE in Fig. 5. It generates test cases according to $R6$ and $R7$. The algorithm starts with a graph G and transforms into graph G' such that the set of source–sink paths in G' is the same as the set of source–sink paths in G with criterion $R6$. (We do not discuss criterion $R7$ but it is easy to modify the procedure to generate all simple cycles to satisfy $R7$.)

The intuition of the algorithm is to delete all white edges except those needed to reach black transitions from the source node (or from the black transitions to the sink nodes). Line 1-3 maintains a marker 'super-edge' for each source node to black transition path. On line 10, we replace this marker with the shortest path. Lines 4-6 perform an analogous function for sink nodes.

BASIC-COVERAGE

Input. transition graph G of a FSM with a designated source state.
Output. test generation according to $R6$.

```
1  for each node v that has an outgoing black edge
2      if there is a path from the source node to v then
3          add a super-edge from the source node to v;
4  for each node v that has an incoming black edge
5      if there is a path from v to a sink node then
6          add a super-edge from v to this sink node;
7  delete all white edges from the graph;
   /* the only remaining edges are super-edges or black edges */
8  generate all acyclic paths in the resulting graph with the
   modification that no super-edge should follow or precede a
   white edge;
9  replace each super-edge with the shortest path in the
   original graph; /* this may introduce some loops */
10 process each path to get rid of loops;
```

Fig. 5. Generating acyclic paths for basic coverage(according to $R6$)

Proposition 1: Algorithm BASIC-COVERAGE in Fig. 5 constructs all acyclic paths according to criterion $R6$.

Discussion: Both Algorithms BASIC-COVERAGE and ADEQUATE-COVERAGE are useful. If automated test execution is available then Algorithm ADEQUATE-COVERAGE is the preferred algorithm. If on the other hand, test cases have to be executed manually then Algorithm BASIC-COVERAGE is likely to produce a manageable set of test sequences. Alternatively, one can always start with Algorithm BASIC-COVERAGE since it captures the most critical interoperability behavior. If the system passes tests generated by Algorithm BASIC-COVERAGE, one can move to a broader coverage provided through Algorithm ADEQUATE-COVERAGE.

IV. ITIS: AN INTEROPERABILITY TEST GENERATION SOFTWARE TOOL

The algorithms described in earlier sections are part of the ITIS (Interoperability Testing Intelligent System) project at Lucent Bell Laboratories. ITIS is a software tool for automated interoperability test generation. It is inexpensive and completely portable. ITIS is written in ANSI C and Tcl/Tk. We first describe its architecture and then report experimental results.

A. Architecture of ITIS

ITIS has a GUI for user input and for displaying test sequences. The workflow of ITIS is shown in Fig. 6. The input to ITIS is an EFSM description of the system behavior, of which we have complete information. ITIS first conducts a reachability analysis to convert the EFSM into an FSM, and then applies different algorithms discussed in previous sections to generate test sequences.

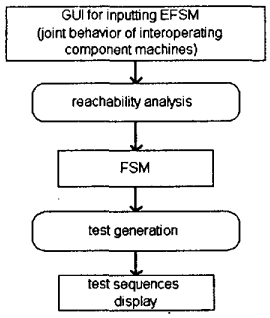

Fig. 6. The workflow for interoperability test generation

B. Experiments

We have used ITIS to generate the interoperability test cases for End-user VoIP testing. The EFSM for this protocol is shown in Fig. 7 in Appendix. This EFSM has 21 states and 68 transitions, among them 24 transitions are colored as black, others white. The shrunk DAG generated by ITIS from this EFSM contains 3 SCC nodes, only one of them contains more than one state. The number of generated test sequences by applying different algorithms is shown in the following table.

TABLE I
COMPARISON OF THREE ALGORITHMS

Algorithms	loop-free paths	loops	final tests
EXHAUSTIVE-COVERAGE	950	424	1752
ADEQUATE-COVERAGE	508	424	908
BASIC-COVERAGE	16	4	22

The 22 test sequences generated by Algorithm BASIC-COVERAGE are in Appendix. The coverage of these test sequences is complete with respect to the requirements for the system modeled by the EFSM. The requirements appear in Bellcore's LATA Switching Systems Generic Requirements [1], which includes hundreds of requirements on the behavior of a telephone switching system. These include many, such as trunking requirements and power requirements, that are not relevant to Internet telephony and are not modeled by the EFSM. However, there are 40 requirements that are applicable to Internet telephony. All of these are tested by the 22 test sequences generated by Algorithm BASIC-COVERAGE. The detailed descriptions of the test generation and the experiments on H.323 are reported in [8].

V. CONCLUSION

Heterogeneity is one of the prominent features of networking systems, and interoperability is ubiquitous and has become a major hurdle for system reliability and quality of service. Interoperability testing is indispensable for the integration of reactive systems. Conventional conformance testing techniques do not apply since we want to test the system interfaces rather than to check implementations vs specifications.

We propose several algorithms for automatic generation of interoperability testing cases, incorporating a range of coverage and redundancy criteria. The tests generated are complete and irredundant. They are applied to VoIP interoperability testing, and provide promising results.

REFERENCES

[1] LATA switching systems generic requirements: Call processing. Technical Report GR-505-CORE, Bellcore, December 1997.

[2] G. Bonnes. IBM OSI interoperability verification services. In *IFIP TC6 WG6.1 The 3rd International Workshop on Protocol Test System*, 1990.

[3] R. Castanet and O. Kone. Deriving coordinated testers for interoperability. In O. Rafiq, editor, *Protocol Test Systems, VI(C-19)*, pages 331–345. Elsevier Science Publisher B. V.(North-Holland), 1994.

[4] J. Gadre, Rohre C, C. Summers, and S. Symington. A COS study of OSI interoperability. *Computer Standards and Interfaces*, 9(3):217–237, 1990.

[5] R. Hao. *Protocol Conformance and Interoperability Testing based on Formal Methods.* PhD thesis, Tsinghua University, P.R.China, 1997.

[6] S. Kang and M. Kim. Test sequence generation for adaptive interoperability testing. In *Proceeding of Protocol Test Systems, VIII*, pages 187–200, 1995.

[7] S. Kang and M. Kim. Interoperability test suite derivation for symmetric communication protocols. In *Proceeding of FORTE/PSTV'97*, 1997.

[8] Bell Labs. Interoperability testing intelligent system. http://www-zoo.research.bell-labs.com/ nr/11341/it-is/.

[9] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - a survey. *The Proceedings of IEEE*, 84(8):1089–1123, August 1996.

[10] G. Luo, G. Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized wp-method. *IEEE Transactions on S.E.*, 20(2):149–162, 1994.

[11] N.Arakawa and T. Soneoka. A test case generation method for concurrent programs. In R.J. Heijink J. Kroon and E. Brinksma, editors, *Protocol Test Systems, IV*, pages 95–106. Elsevier Science Publisher B. V.(North-Holland), 1992.

[12] O Rafiq and R. Castanet. From conformance testing to interoperability testing. In *The 3rd International Workshop on Protocol Test Systems*, 1990.

[13] G. S. Vermeer and H. Blik. Interoperability testing: Basis for the acceptance of communicating systems. In *Protocol Test Systems, VI(C-19)*. Elsevier Science Publisher B. V.(North-Holland), 1994.

APPENDIX

I. TEST SEQUENCES GENERATED BY BASIC-COVERAGE

1. Off-hook B /DialTone B, Off-hook A /DialTone A, Dial A B /LineBusyTone A, On -hook A /, On-hook B /
2. Off-hook B /DialTone B, Dial B A /AudibleRinging B,Ringing A, Off-hook A /, On-hook A /, On-hook B /
3. Off-hook B /DialTone B, Off-hook A /DialTone A, Dial B A /LineBusyTone B, On -hook B /, On-hook A /
4. Off-hook A /DialTone A, Dial A B /AudibleRinging A,Ringing B, Off-hook B /, On-hook A /, On-hook B /
5. Off-hook B /DialTone B, Dial B A /AudibleRinging B,Ringing A, Timeout AB /, Off-hook A /DialTone A, Dial A B /LineBusyTone A, On-hook B /, On-hook A /
6. Off-hook B /DialTone B, Off-hook A /DialTone A, Dial A B /LineBusyTone A, Di al B A /LineBusyTone B, On-hook A /, On-hook B /
7. Off-hook A /DialTone A, Dial A B /AudibleRinging A,Ringing B, Timeout AB /, On-hook A /
8. Off-hook A /DialTone A, Dial A B /AudibleRinging A,Ringing B, Timeout AB /, Off-hook B /DialTone B, Dial B A /LineBusyTone B, On-hook B /, On-hook A /
9. Off-hook B /DialTone B, Off-hook A /DialTone A, Dial B A /LineBusyTone B, Di al A B /LineBusyTone A, On-hook A /, On-hook B /
10. Off-hook B /DialTone B, Dial B A /AudibleRinging B,Ringing A, On-hook B /
11. Off-hook B /DialTone B, Dial B A /AudibleRinging B,Ringing A, Timeout AB /, On-hook B /
12. Off-hook B /DialTone B, Dial B A /AudibleRinging B,Ringing A, Off-hook A /, On-hook B /, On-hook A /
13. Off-hook A /DialTone A, Dial A B /AudibleRinging A,Ringing B, Off-hook B /, On-hook B /, On-hook A /
14. Off-hook A /DialTone A, Dial A B /AudibleRinging A,Ringing B, Onhook A /
15. Off-hook A /DialTone A, Dial A B /AudibleRinging A,Ringing B, Off-hook B /, On-hook B /, Timeout AB /DialTone A, Dial A B /AudibleRinging A,Ringing B, Onhook A /
16. Off-hook A /DialTone A, Dial A B /AudibleRinging A,Ringing B, Off-hook B /, On-hook B /, Timeout AB /DialTone A, Dial A B /AudibleRinging A,Ringing B, Off-hook B /, On-hook B /, Off-hook B /, On-hook B /, On-hook A /
17. Off-hook B /DialTone B, Dial B A /AudibleRinging B,Ringing A, Off-hook A /, On-hook A /, Timeout AB /DialTone B, Dial B A /AudibleRinging B,Ringing A, Off-hook A /, On-hook A /, Off-hook A /, On-hook B /, On-hook A /
18. Off-hook B /DialTone B, Dial B A /AudibleRinging B,Ringing A, Off-hook A /, On-hook A /, Timeout AB /DialTone B, Dial B A /AudibleRinging B,Ringing A, Timeout AB /, On-hook B /
19. Off-hook B /DialTone B, Dial B A /AudibleRinging B,Ringing A, Off-hook A /, On-hook A /, Timeout AB /DialTone B, Dial B A /AudibleRinging B,Ringing A, On-hook B /
20. Off-hook A /DialTone A, Dial A B /AudibleRinging A,Ringing B, Off-hook B /, On-hook B /, Timeout AB /DialTone A, Dial A B /AudibleRinging A,Ringing B, Timeout AB /, On-hook A /
21. Off-hook A /DialTone A, Dial A B /AudibleRinging A,Ringing B, Off-hook B /, On-hook B /, Timeout AB /DialTone A, Dial A B /AudibleRinging A,Ringing B, Off-hook B /, On-hook B /, Off-hook B /, On-hook A /, On-hook B /
22. Off-hook B /DialTone B, Dial B A /AudibleRinging B,Ringing A, Off-hook A /, On-hook A /, Timeout AB /DialTone B, Dial B A /AudibleRinging B,Ringing A, Off-hook A /, On-hook A /, Off-hook A /, On-hook A /, On-hook B /

II. EFSM FOR END-USER VOIP TESTING

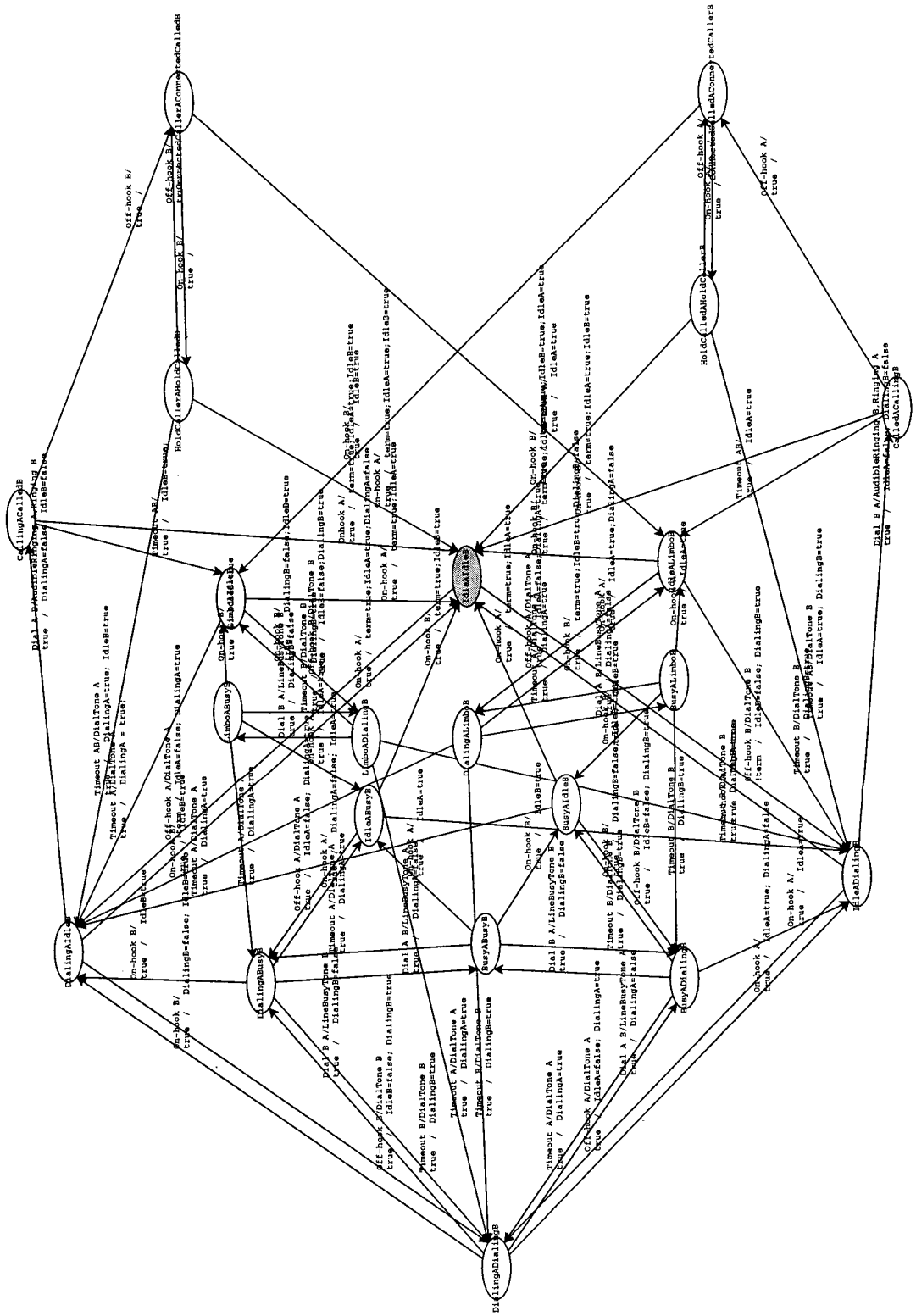

Fig. 7. EFSM for End-user VoIP testing