Formal Specification, Verification, and Automatic Test Generation of ATM Routing Protocol: PNNI

David Cypher[a], David Lee[b], Marta Martin-Villalba[a], Christiaan Prins[a], and David Su[a]

[a]National Institute of Standards and Technology, Gaithersburg, Maryland

[b]Bell Laboratories, Lucent Technologies, Murray Hill, New Jersey

Abstract

This paper presents an effort to use formal tools to model, validate, and generate test suites for the ATM network routing protocol called Private Network-Network Interface (PNNI). PNNI consists of three layers of protocols: the Hello protocol for identifying the status of NNIs; the Database Synchronization protocol for maintenance of routing databases; and the Peer Group Leader Election protocol for operations of hierarchical routing. Parameterized extended finite state machine (EFSM) models were developed for each protocol using the PROMELA language; communicating EFSM's are used to model all PNNI protocols. The models were executed using the SPIN simulator, which performs protocol validations and generates reachability graphs. It simulates two network nodes communicating with each other. The reachability graph for the Hello protocol model was produced and fed into an automatic test case generation tool, PITHIA, to produce test cases. This paper discusses some of the problems encountered while implementing the models and their solutions, such as state space explosion and lack of modeling power of timers in PROMELA.

Keywords

Modeling, Protocol Verification, Conformance and Interoperability Testing, Test Suite Generation

1 Introduction

For the reliability of ATM networks, there is an urgent need to apply some formal methods in validating these protocols and in generating test cases. The tools needed to accomplish these purposes include: a technique to model the protocol, a formal description language to describe the model, a protocol simulator to perform verifications, and a test generator to produce test cases. This work is a case study of applying formal verification and test generation tools to the ATM network routing protocol, or the ATM Forum Private Network-Network Interface (PNNI) Specification Version 1.0 [1].

The PNNI specification consists of three layers of protocols: the Hello protocol for identifying the status of NNIs; the Database Synchronization protocol for maintenance of routing databases; and the Peer Group Leader Election protocol for operations of hierarchical routing. Each of the PNNI sub-protocols can best be modeled as communicating Extended Finite State Machines (EFSM) with parameters [2]. The system behavior of the PNNI protocol system is the combined effects of three communicating EFSMs. The issue at hand is: Is it necessary to develop a complete composite state machine to correctly model PNNI, and if so, how? Furthermore, since a PNNI system represents an ATM network switching node, within a network switching nodes are expected to interact with each other. Therefore what is the minimum number of nodes needed to correctly simulate the operation of PNNI?

In our study, we take two nodes connected by a full-duplex channel, based on the following observations. We want to detect design/specification errors from verification, implementation errors from conformance testing, and errors from both the design and implementation from interoperability testing. Suppose that an error is revealed using a two-node model, then obviously, faults in the specification are detected. Conversely, suppose that there are faults in PNNI in a multiple node network. Since it is a hierarchical protocol, we can group the nodes into two peer groups. Each peer group then behaves like a single node. These two connected peer groups behave like our previous two nodes, so the faults will manifest in this two-node network as well. Therefore, we confine ourselves to a two-node network for our verification and testing study.

The tool set we use is PROMELA, SPIN [3], and PITHIA [4] for modeling, simulating, and generating tests for the protocols, respectively.

The rest of this paper is organized as follows: section 2 presents a brief introduction of the PNNI protocols; section 3 presents the development of the models, processes and results of simulations using SPIN; section 4 describes the process of test generation from the reachability graph; and section 5 presents concluding remarks.

2 Private Network-Network Interface 1.0

The Private Network-Network Interface (PNNI) is a set of protocols that define the dynamic distribution of routing information and the signaling used, once the routing information is complete. These routing protocols are a derivation of protocols used in the Internet. The signaling protocols are not studied in this project.

The routing protocols dynamically obtain, distribute, and summarize information about the nodes (i.e. ATM switches) and the links connecting these nodes. A hierarchy mechanism is used to reduce the amount of information that has to be distributed. This hierarchy mechanism is built upon groupings of switches into peer groups. Within a peer group all switches share the same routing information. A peer group can be viewed by other switches as a single logical switch. This single logical switch can itself be a member of another higher level peer group. This process continues, thus building a hierarchy.

There are three main protocols for accomplishing the task of routing. They are the Hello Protocol, the Database Synchronization Protocol including Flooding, and the Peer Group Leader Election (PGLE) Protocol. These protocols all interact.

2.1 Hello Protocol

The Hello protocol's purpose is to discover and to verify the identity of the neighboring node at the other end of a physical link or virtual path connection, as well as to maintain the status of its connectivity to that neighboring node. The information learned about the other node includes its ATM address, node identity, port identity, peer group identity. Other information is also passed but is not used immediately by the Hello protocol.

The Hello protocol can be divided into two parts: an inside part where the two neighboring nodes are within the same peer group, and an outside part where the two neighboring nodes are in different peer groups. For the latter there are two further possibilities. The two neighboring nodes may either share a common higher level peer group or not. Two neighboring nodes that are in different peer groups are called border nodes.

The Hello protocol is the first protocol to run once a link between two neighboring nodes is active. The Hello protocol operates on a per link basis.

2.2 Database Synchronization and Flooding

The Database Synchronization protocol's purpose is to distribute the currently known routing information between two neighboring nodes that are in the same peer group. When the database synchronization process is complete both neighboring nodes will have exactly the same routing information. The Flooding procedures are the necessary extensions to the database synchronization protocol in order to keep all routing information within a single peer group current.

The Database Synchronization protocol is started with the event *AddPort*, which is generated when the first link between two neighboring nodes running the Hello protocol enters the state, Two Way Inside. The synchronization of the databases is done through the exchange of Database Summary (DS) packets. These describe the PNNI Topology State Elements (PTSEs) that a node has in its database. In response to these packets a node requests the PTSEs that it does not have from its neighbors. The other nodes respond to these requests by sending the appropriate PTSEs in PNNI Topology State Packets (PTSPs). After these exchanges the databases will be synchronized.

This protocol runs between a pair of neighboring nodes, not on a per link basis and thus can involve more than one link. The two neighboring nodes form a Master/Slave relationship for the distribution of the routing information.

2.3 Peer Group Leader Election

The Peer Group Leader Election protocol provides a mechanism for choosing one node from within a single peer group to represent the other nodes as a single logical group node. This elected node, or Peer Group Leader (PGL) will then perform other functions for building a hierarchical network topology.

Normally all the nodes of the peer group participate in the peer group leader election. Each node is configured with a peer group leadership priority value, which indicates its desirability to become PGL. There are two configurations for a node. Either the node does not want to be PGL and advertises a zero value or the node wants to be considered for PGL and advertises a non-zero value.

2.4 Other protocols

There are other protocols defined within the PNNI v1.0 specification, but we are not pursuing them at this time, as most implementations are not using them.

3 PROMELA and SPIN

PROMELA is the formal specification language that we used to describe our validation models. SPIN is a software tool for simulating the behavior of validation models written in the PROMELA language. Both of these are described in [3].

3.1 Limitations of the PROMELA language

There were a number of limitations encountered when using PROMELA to model the PNNI protocols. The first was the lack of a notion of time. The second was the difficulty in encoding the large number of variables, messages and data structures. All of these deal with defining an abstract model as a formal model of the protocol, and not as an implementation. The major difference between a model and an implementation of a protocol, is the level of detail. A model is to provide high level abstraction and is only expected to prove that the protocol is well specified (i.e. behaves as expected).

PROMELA is a formal modeling language that has no support for time-related commands. Since the three protocols use timers for generating timeout events, this creates a serious problem. There are several ways to model the timers in PROMELA.

- Using the PROMELA *timeout* statement. Actually this is a Boolean read-only variable that becomes true when no statements can be executed in any of the active processes and which is false in all other cases.
- Using a selection that is always possible by using a guard that is always true. In this way a timeout is possible to occur at any moment, even when the neighboring node is active.
- Using a counter, which counts how many times a certain event occurs successively (for example: re-sending of a Hello message). When the counter reaches a certain value, the timer is considered to expire.
- Indication of a timer expiry through a *channel*. The best way of indicating the expiration of the timer is the use of a *channel* instead of using global variables.

This must be done whenever the real timer depends on a certain event or a number of events that occur in another process, or when a timer has to be "faked" by an outside dummy process firing it randomly.

The PROMELA statement *timeout* (first way) makes an event occur whenever there is no other possible action to take. This timeout is global, i.e. the timeout would only occur whenever there is no possible event in any running process. If both nodes are ready to fire a timer expiry at the same time, PROMELA randomly chooses the node that will fire it. The fact is that a timer of a node should be independent of the timers of the other node, so this way of modeling is not valid for simulation purposes. Also, in some cases a timer expiry not only has to occur when there is no other possibility, but is supposed to occur every certain interval. Although simulation results are not representative of real-time behavior, using the *timeout* statement for modeling a timeout is no problem in verification, since the sequence of messages (action taken after timer expiry) is still correct.

Using an always possible selection (second way) may result in a model that describes the protocol better, but has some undesirable side-effects. One side effect is that it may cause undesirable behavior during the simulation because events occur randomly. Another side-effect is the resulting large state space.

Also the use of a counter as a simulation for timeout (third and fourth way), will normally result in a state space explosion.

PROMELA requires the use of *channels* to communicate between processes. These *channels* are defined for fixed format messages. This created problems since all of the protocols send messages that are variable in size and content. The hello packet contents are especially complicated because they must be examined and compared to a local data structure in order to determine the event received. To create as much of the protocol behavior as possible with the limitations we faked most of the fields and created formats for PROMELA's use that would never occur in a real environment. For the Database Synchronization and PGLE the number of variables (i.e. stored routing information) is critical for the behavior of the protocols. However, in order to keep the state space from exploding, we only modeled a minimum number. The size of the models created became a limitation when we used SPIN.

3.2 General modeling

In the original PNNI specification each of the three PNNI protocols has a corresponding Finite State Machine (FSM). We modeled each of these protocols separately, using the PROMELA language, as EFSM models. Next we tested each of the protocols, except for PGLE, by connecting two instances of the modeled protocol. In this way we are able to simulate the interoperation of the protocol between two nodes. The configuration is shown in Figure 1. The PGLE was tested as a single node where the database information was changed instead of sending events. In the remainder of this paper, the terms FSM and EFSM are interchangeable.

PROMELA requires an *init* process that starts all other processes. The *init* process initializes the *channel(s)* between two instances (i.e. processes) of the particular protocol's FSM.

Our models consider only packet losses, since these are the most common error in a real ATM network environment. Other errors, like bit-flips, that the protocol is capable of handling, were not modeled.

3.3 Model for Hello

The Hello protocol was modeled in PROMELA as processes representing the Hello EFSM, communicating through *channels*.

3.3.1 General Structure

Basically the model consists of three processes and a link. Two of the processes are the Hello EFSMs that are connected with a link and an *init* process that combines these two processes to operate together (Figure 1). The link consists of two *channels*,

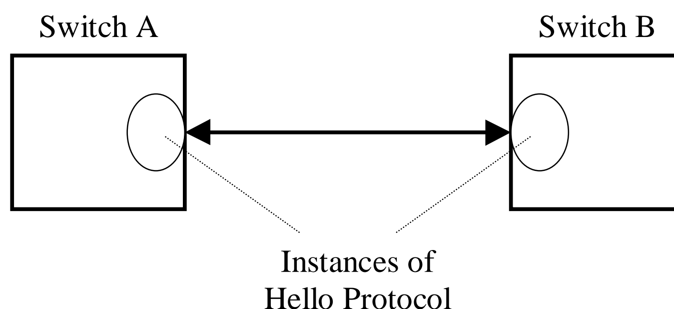

Figure 1: Model for Hello

one in each direction. At startup the two processes receive a LinkUp message from the *init* process, thereafter they begin exchanging Hello packets.

The Hello protocol defines two timers: the Hello and Inactivity Timers. The expiry of the Hello timer was modeled using the predefined *timeout* statement. However we modeled the expiry of the Inactivity Timer using a counter. The number of times a Hello timeout occurred and the packet was lost, was counted until a certain value was reached, upon which the timeout would occur.

Three configurations were used to validate the Hello model. In the first configuration, the two nodes were in the same peer group. In the second and third configurations, the two nodes were configured to be in different peer groups, one with a common higher peer group and the other without.

3.3.2 Limitations and assumption

The Hello protocol allows variable number of hierarchical levels. Since PROMELA allows only fixed length messages, we implemented a fixed three level network topology. The format checking required by the protocol was not simulated.

3.3.3 Results of Simulation and Verification

No deadlocks or live-locks were found in the model. The lines of code that were not reachable were all explainable. Therefore we conclude that the Hello protocol is complete and correct. Different than verification, testing may require a complete reachability graph of the two-node network, communicating EFSM's via a full-duplex channel, a graph with a large number of states. For clarity, in this report, we consider two simplified Hello Models.

3.3.4 Simplified Hello Model

We now discuss a simplified model of the Hello protocol. For clarity we will use this model to illustrate various concepts and analysis.

The main change in this model from the previous model is the absence of variables. The first step for creating this model was deleting every data structure and every local and global variable. Then some behavior had to be added to keep the model correct.

In this model the first step in the simulation is to configure the nodes to be both in the same peer group (the inside part of the FSM), or configure both nodes to be in different peer groups (the outside part of the FSM). In this model, the events are just messages received through a *channel* declaring which event is occurring. This in contrast with the first model, where most of the events were guards composed of different variable values that had to be checked. In the first version the guard of the event *1-Way Inside Received* was: *(Remote.peergroupid==Home.peergroupid && Remote.r_portid==0 && Remote.r_nodeid==0).* In the new version this is: *in?OneWayIn.* The latter method can simulate only the sequence after *OneWayIn* is successfully received, not the sequence where the guard in the former case fails.

This model covers some behavior that is not simulated in the original version. In this model it is possible that the node changes its configuration, at some point in the simulation. It is possible that the initial configuration, where both nodes are in different peer groups but with a common higher level peer group, changes to a configuration, where both nodes are in different peer groups but with no common higher level peer group and vice versa. So this model makes two transitions possible, which were not present in the original version.

The final simplified model was constructed using these characteristics:

- the event Hello Timer expired is simulated using the *timeout* statement,
- the event Inactivity Timer expired is simulated by receipt of a message from the other process,
- the *channels* for interchanging Hello messages have a queue of size one, and
- the *channels* that fire the event Inactivity have a queue of size one.

3.4 Model for Database Synchronization and Flooding

The Database Synchronization and Flooding protocols were modeled together in PROMELA as one EFSM. There are two processes and four *channels* (per direction) to simulate two nodes as shown in Figure 2.

3.4.1 General structure

For the model, we assume the following situation. The two neighboring nodes must be in the same peer group. They communicate through one link, using only one port per node. The assumption is that at startup both nodes have different topology information in their database, though, some information elements can be present in both (i.e. there can be overlap). The database will contain information about nodes in the peer group even though they are not present in the actual model. This makes the verification easier because the behavior of two nodes in a whole peer group can be simulated using only two nodes. This has some implications. First, PTSEs can only be requested from one neighbor, namely the one on the other side of the link and not

from any other neighboring node. Furthermore, real flooding of PTSEs is obviously not possible with only two nodes. Nevertheless, the major part of the Flooding protocol is still covered by this case.

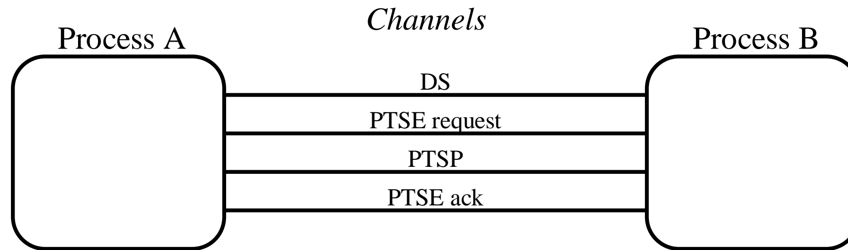

Figure 2: Model for Database Synchronization and

By only modeling one link between the two neighboring nodes, some events defined in the Database Synchronization protocol can not occur or can only occur once. Because PROMELA does not have the capability for more than one format of messages to be sent over one *channel*, every type of packet (i.e. DS, PTSE request, PTSP and PTSE acknowledgement) must have a separate *channel* defined. The PTSE request and PTSE acknowledgement *channels* have a buffer size of one, since the procotol is a lock-step mechanism. The DS and PTSP *channels* must have a buffer size of two, since multiple messages of this type may be outstanding.

Three configurations of the topology database were defined to cover all situations. These configurations cover all the major parts of the model. A simulation was run for each of these configurations. During the simulations (and also verification) the size of the DS, PTSE request and PTSE acknowledgement packets was limited to one set containing two PTSEs (or PTSE identifying information).

3.4.2 Limitations and assumptions

- Creation of new information (i.e. PTSEs) is not modeled, since this would make the model even more complicated. The result is that part of the flooding protocol is never used. Since all handling of PTSEs in Database Synchronization is done by Flooding, the major part of Flooding is used.
- Furthermore, the PTSEs in the database are not being aged, so they are static. Otherwise, this would also make the model unnecessarily complex.
- A DS packet contains the More bit. In the model the selection is made to make this bit zero when the DS packet has contents and is the last in the sequence and/or if the DS packet is empty.
- Finally, when in Database Synchronization a node receives a PTSP eventually an acknowledgement packet is transmitted, although this is not necessary for the operation of the protocol. The use of the PTSE request list makes sure that the PTSEs are requested until they are received from the neighbor. So, there is no need for acknowledgements. Although this operation is not necessary it is copied from the protocol specifications for compliance.

3.4.3 Results of Simulation and Verification

During verification of the model the three configurations proved to sufficiently cover the model. Also the packets in the model had a maximum size of one set containing two PTSEs (or PTSE identifying information). During test-verifications it was seen

that this gives the best results without doing numerous verification-runs with different sizes of the packets.

The SPIN tool was used to do the verifications. A supertrace/bitstate verification is done. In this mode, there is no assurance of complete coverage, but with the amount of memory that is used this coverage on average is at least 99.9 percent. There is no check for weak-fairness, assertions, or cycles.

No deadlocks or live-locks were found. The lines of the code that were not reachable were all explainable. The verifications showed that the model has no essential code that is not reached. No invalid end states were detected.

3.5 Model for Peer Group Leader Election

3.5.1 General Structure

The Peer Group Leader Election model consists of three processes: the EFSM for the PGL Election process that provides the PGL Election protocol in a node; the Simulation process that fires the events corresponding to the interactions between the PGL Election protocol and the Hello, Database Synchronization and Flooding protocols; the *Init* process that combines these two processes together. The interactions between the PGL Election protocol and the Hello, Database Synchronization and Flooding protocols were simulated instead of using the already defined models.

The PGLE model did not use the same configuration as the other two models. This model used a *channel* of size zero, which defines a rendezvous port that can only pass and not store. Message interactions via these ports are synchronous. The rendezvous communication is binary, so there must be only one process sending over the *channel*, in this case the Simulation process, and only one process reading, that is the PGL Election process. A global variable was needed to avoid a race condition, since this model uses a global database that is accessed by two different processes (Simulation and PGL Election). This global variable allows the Simulation process to modify the database without any interleaving from the PGL Election process and vice versa.

3.5.2 Limitations and Assumptions

- Again due to the lack of timing in PROMELA, the InitDelay state has not been modeled, since its only use is to make the node wait for a certain amount of time. In consequence, the PGLInit Timer is not used.
- The OverrideUnanimity and ReElection Timers are modeled using a guard that is always possible (i.e. these timers can be fired at every time in the corresponding states).
- No PTSEs are sent from the node. For simplification, whenever the Node is supposed to send a PTSE advertising new information, the information is changed directly in the database.

- The database was set up for the information of four nodes. These are: the node that runs the PGL Election FSM, called N, two nodes with lower priorities than N, called L1 and L2; and another with a higher priority than N, called H. This

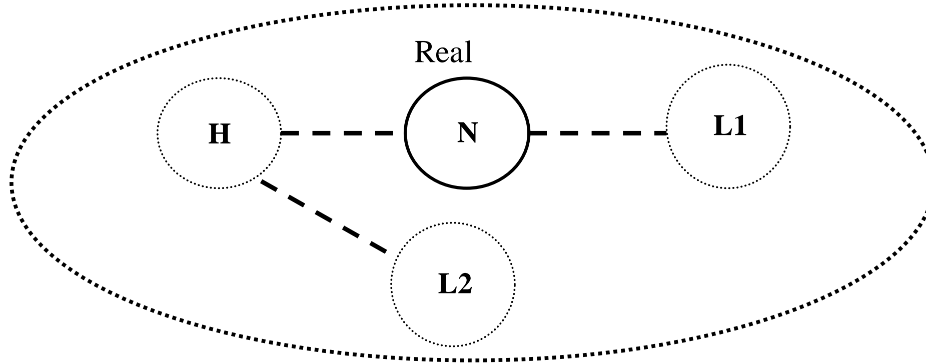

Figure 3: Network topology being used in the PGLE model.

configuration is shown in Figure 3. At the beginning of the simulation, the nodes L1, L2 and H are not reachable from the node N.
There are four different configuration possibilities:
- N is alone in the peer group, it does not find any neighbor.
- N recognizes as a neighbor the node L1. As L1 has lower priority than N, the node N will elect itself as Preferred PGL.
- N recognizes as a neighbor the node H. As the node H has a higher priority than N, N will decide that it is not its Preferred PGL.
- N recognizes as neighbors the nodes L1 and L2. In the model L2 elects the node H as its Preferred PGL, even if H is not reachable. So the election result will be that N got two thirds of the votes, and when the OverrideUnanimity timer fires, N will elect itself as PG Leader.

If no node in the peer group has a non zero priority, then no PG Leader is elected. Currently, this case is not modeled. One of these four possibilities is selected randomly when the event DBReceived is fired. The idea is that with these four different configurations all the states and events of the FSM are reached.

3.5.3 Results of Simulation and Verification

Two verifications of the model have been done, one checking for invalid end states and another checking for the existence of live-locks and deadlocks. A default partial order reduction algorithm and the supertrace/bitstate search have been used. Both of the verifications were successful, no live-locks, deadlocks or invalid end states were detected. The state-space is not too big, around sixty thousand unique global states. The expected coverage of the search on average is higher than 99.9 percent. There is no unreachable code.

4 Automatic test generation: PITHIA

We now discuss conformance and interoperability testing of PNNI.

Conformance testing checks whether an implementation of PNNI protocol conforms to the specification. Much work has been done [2] on conformance testing and we do not intend to provide a survey here.

Interoperability testing is becoming an important topic in protocol analysis. Systems may fail to interoperate for various reasons: (1) Implementations do not conform to the specifications, resulting in performance degradation or system failures; (2) Design errors may be revealed only when systems interoperate with other systems; (3) A combination of design and implementation errors that cause serious system faults. In the interoperability testing literature, the term "interoperability" is often defined differently. Rather than debate the definition of interoperability, we take the following approach.

4.1 A Case Study: Hello Protocol

As a case study, we use the Hello protocol with a two-node network model. Each node (process) contains an implementation of the Hello protocol. Each test sequence consists of an execution sequence (often called scenario), starting with both nodes in the initial state. We want to construct test sequences so that the specification code of each node protocol is exercised at least once. Furthermore, we want to minimize the number of tests. The resulting set of tests could be served as conformance tests when they are projected into a single node protocol; each implementation code line is exercised at least once, and this is a commonly used criterion for conformance testing. On the other hand, since each implementation code line of both nodes is executed jointly through exchanging messages via the full-duplex channel, the combined behaviors of the two nodes are also tested. This will also reveal certain design and implementation errors. Note that given the size of the system a complete testing of the structural equivalence of the specification and the implementation using a checking sequence [2] is impossible.

4.2 Test Sequences and Covering Paths of the Reachability Graph

We construct test sequences based on the reachability graph obtained from SPIN. The initial node of the graph contains the initial state of the two processes (nodes of the same Hello protocol) along with the initial variable values and the channel content (empty).

Similarly, each node in the graph consists of two states, one for each process, and the current channel content. There are edges between nodes. Each edge corresponds to a transition between two states of a process, and each transition corresponds to a specification code line. Therefore, each edge is uniquely identified by a process and a specification code line of the process, denoted by a distinct color.

In summary, we have a reachability graph with an initial node and edges between nodes. Each edge has a color; a same color may appear at multiple edges. We want to find a set of paths (tests) such that each color is covered at least once.

4.3 A Greedy Method

Our goal is to generate a set of paths such that each color of the reachability graph is covered at least once. Since the same color may appear at more than one edge, it is a NP-hard problem to find a minimal number of such covering paths [2]. We apply the following greedy method.

We first shrink each strongly connected component (SCC) into one node, which contains all the colors in the SCC. We have a directed acyclic graph. We topologically sort the nodes in the graph and process them bottom-up. Suppose that we are processing node U with descendants $V_1,...,V_k$, each of which has been processed, i.e., we have computed an optimal path from V_i that covers a maximal number of colors where i=1,...,k. We want to find a path from U, i.e., to determine an edge among (U,V_i), i=1,...,k. We select the node V_i such that the combined number of colors of edge (U, V_i) and the chosen path from V_i to a sink node is maximal. We can process all the nodes bottom-up until we reach the initial node, and we obtain a path that supposedly covers a maximal number of uncovered colors. We delete all the colors covered, and construct another path from the initial node until all the colors are covered.

The detailed method is described in [4]. It performed well on practical systems [2].

4.4 Tests Generation Using PITHIA

We apply PITHIA to both the simplified and original versions of the Hello protocol. As indicated earlier, we consider a two-node network, i.e., two processes of the Hello protocols that are connected by a full-duplex channel.

For the simplified version, the reachability graph contains 6,079 nodes and 7,584 edges. It is a sparse graph of 1,850 SCC's and most of them are singleton node SCC's. Five tests cover all the specification code lines of the two jointly operating processes.

For the more complex original version, the reachability graph contains 60,230 nodes and 68,140 edges. It is a sparse graph of 8,122 SCC's and most of them are singleton node SCC's. Five tests cover all the specification code lines of the two jointly operating processes.

5 Concluding Remarks

We have developed EFSM models of the ATM network routing protocols and verified that the specification of these protocols has no deadlocks, live-locks, or unreachable states. We have generated test sequences that covers the Hello protocol based on its reachability graph. Here are some lessons we learned:

1. Difficulty in transforming from protocol specification to abstract model - The protocols that were modeled were described in terms of both control and data flows. This is in contrast to that which is needed for use in existing tools. Existing tools rely heavily on control flows only, not data flows. Therefore the need to convert the data flows in the specification into control flows for use in the tools is difficult at best. The tools can accommodate data flows, but the result is very cumbersome and creates large state spaces.

2. Difficulty in working with a practical protocol - The PROMELA and SPIN tools are very powerful. It still may not be feasible, however, to work on real protocols that are relatively complicated. Major problems were state space and size of output for reachability graphs. We did not attempt to do a combined EFSM model of the three protocols, Hello, Database Synchronization, and PGLE as planned.

3. Handling of timers in PROMELA - The *timeout* feature in PROMELA speeds up the simulation, as any time the model has nothing to do, timeout is implied. This imposes a requirement, however, that a timeout will occur only when both communicating EFSMs have nothing to do. This makes simulation of independent timers, or asynchronous timeouts between models difficult, if not impossible. The SPIN execution environment also makes it impossible to simulate the case where two models generate messages independently due to timeouts. There are other situations such as multiple timers, or periodical timers, which may fire even if there are other events going on.

4. Interoperability testing and state explosion. - For interoperability testing, we need to cover a large reachability graph so that all the specification code lines of the two Hello protocol processes connected by channels can be covered with their joint executions. With the size of PNNI, it pushes the memory and processing time to the limit. Space and time efficient algorithms are needed for the test generation. Surprisingly, the number of tests needed is small: five. Therefore, only five resets are needed for both conformance and interoperability testing.

Acknowledgment. We are deeply indebted to Gerard Holzmann of Bell Labs for spending long hours in consultations and in helping us running some simulations. Without his help, we would not have accomplished the work.

6 References

[1] The ATM Forum Technical Committee Private Network-Network Interface Specification Version 1.0 (PNNI 1.0), af-pnni-0055.000, March 1996.

[2] David Lee and David Su, “Modeling and Testing of Protocol Systems,” *Testing of Communicating Systems*, pp. 339-363, Vol. 10, 1997.

[3] Gerard Holzmann, *Design and Validation of Computer Protocols*, Prentice-Hall, 1991.

[4] David Lee and Mihalis Yannakakis, “PITHIA – an Automatic Test Generation Software Tool for Communication Systems,” Bell Labs. Technical Report 1998.