Applying Formal Methods to a Protocol Standard and its Implementations

Glenn Bruns and Mark Staskauskas
Bell Labs, Lucent Technologies
1000 E. Warrenville Rd.
Naperville, IL 60566, USA
{grb,markstas}@bell-labs.com

Abstract

We explore the use of formal methods in the analysis of the SONET Automatic Protection Switching (APS) protocol. Unlike most protocol studies, we look at both conformance and interoperability aspect of APS. We use the Concurrency Workbench of North Carolina [9] to show proper interoperation of network elements running the protocol, and use the VeriSoft tool [12] to show conformance of Lucent Technologies APS implementation to the APS standard. We define a new notion of conformance, identify problems with APS, suggest general improvements to the structure of protocol standards, and suggest how protocol implementations can be written to allow conformance checks to be automated.

1. Introduction

Communication protocols are typically specified in standards documents that attempt to provide all the information needed for manufacturers to produce conformant protocol implementations. A problem with using standards for this purpose is that they are informal, and hence vague and incomplete. Formal methods offer a promising means of specifying protocols in a way that eliminates the imprecision inherent in natural-language standards. However, in attempting to apply formal methods to protocol standards we have found that most standards lack more than just a precise account of the working of the protocol. They also fail to state what properties are expected of a network running the protocol, and what it means for an implementation to conform to the standard. Of course, there is also the problem of applying tools and techniques to perform the analysis.

We have been considering the above issues in our study of the SONET (Synchronous Optical NETwork) [1] Automatic Protection Switching (APS) protocol. SONET is assuming increasing importance as a physical layer for fiber-optic networks based on asynchronous transfer mode (ATM) and for trunks carrying telephone traffic worldwide. The APS protocol is instrumental in maintaining the availability and fault tolerance demanded by modern voice and data networks. It allows two network elements to share a redundant communication channel that consists of a pair of fiber-optic lines. Based on the conditions of these lines, the protocol is intended to assure that both switches select the better of the two for communication. Our study of APS has been carried out in collaboration with a Lucent development organization that has been implementing the protocol for Lucent's No. 5 Electronic Switching System (5ESS).

We began our work on APS by defining general criteria that capture what it means for a protocol implementation to *conform* to its specification and for a protocol specification to be *valid*. We then formalized the behavior of APS using a pseudocode representation of the protocol that precisely captures the intent of the natural-language specification in the APS standard. To determine if the APS standard is valid, we specified the intended interoperation of APS network elements in the modal mu-calculus [17, 20] and used the Concurrency Workbench of North Carolina (CWB-NC) [9] to show that a model of APS in the CCS process algebra [18] that was derived from the pseudocode satisfies these properties. To determine whether Lucent's APS implementation conforms to the standard, we translated our pseudocode into C, which allowed us to use the VeriSoft tool [12] to perform an exhaustive check of an APS system consisting of our C implementation in communication with the 5ESS APS implementation. As a second check of conformance we compared the implementations to see if the results of their computations of the selected line are always the same.

The direct contribution of our work is the identification of significant omissions and ambiguities in the APS standard. These problems are especially important in light of the role APS plays in fault tolerance for SONET networks. At a higher level, we have defined a new notion of protocol conformance. Informally, by this notion an implementation conforms to a protocol specification if it interoperates properly with any other implementation that conforms in the common sense. We also suggest improvements to protocol implementations that would allow conformance checks to be automated. Finally, our work confirms existing claims [6] that standards lack certain information needed for their analysis.

The remainder of the paper is organized as follows. Section 2 presents a brief overview of the APS protocol, and Section 3 explains the goals of our analyses of it. In the next two sections, we describe the analyses we performed: Section 4 contains our study of the APS protocol itself using temporal logic and process algebra, while Section 5 details our efforts to determine if the 5ESS APS implementation conforms to the APS standard as embodied in our C implementation of the protocol. Section 6 contains our conclusions.

2. The APS Protocol

The APS protocol is part of the SONET standard [1, 2] for digital transmission over optical fiber. SONET is synchronous, with a message (or *frame*) rate of 8000 frames/second.

The APS protocol improves the reliability of a SONET line through the use of a backup line. Figure 1 shows the architecture of a system that uses APS. Here a *switch* is a piece of SONET line terminating equipment, such as a multiplexer. A switch sends frames over both the primary (*working*) line and the backup (*protection*) line, but reads frames from just one line. The two switches normally read frames from the same line. If a switch detects errors on a line, or receives a technician command instructing it to change lines, it coordinates with the other switch to begin reading frames from the other line. To coordinate the switches send an APS message in every frame. A message from a switch includes the line favored by the switch for reading, and a priority level. The line favored by the switch with the higher-priority messages is the one chosen by both switches for reading.

Bellcore and ANSI have standards [1, 4, 5] for SONET, and the ITU-T has a closely-related international standard [16]. Our work on APS is based on section 5.3 of the Bellcore SONET standard [5]. The standard consists of requirements items along with commentary, tables, and figures. There is no mathematical notation in the standard, and the tables serve mostly as structured text, not as representations of mathematical relations. Figure 2 contains requirement R5-59 of the standard.

The APS standard describes a family of protocols. Here

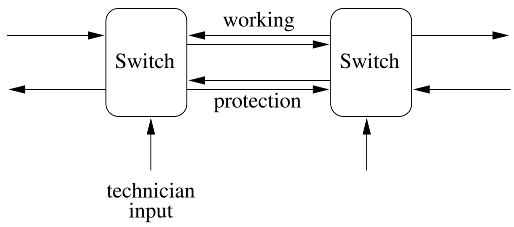

Figure 1. The APS Architecture

[183] In the bidirectional mode, the priorities of the current local request and the remote request on the received K1 byte shall be compared according to the order of priorities in Table 5-3. A received Reverse Request shall not be considered in the comparison, because it assumes the priority of the request to which it is responding. The transmitted K1 byte shall be set to indicate a Reverse Request if any of the following are true:

- The remote request is of higher priority
- The requests are of the same priority, they are higher priority than a No Request, and the transmitted K1 byte is already set to indicate Reverse Request
- The requests are of the same priority, they are higher priority than a No Request, the transmitted K1 byte is not set to indicate Reverse Request, and the remote request indicates a lower channel number

The transmitted K1 byte shall be set to indicate the local request in all other cases.

Figure 2. An Excerpt from the APS Standard

we consider only the linear, bidirectional, nonrevertive, and 1+1 variant. Furthermore, we do not consider the raising of alarms, we assume line error conditions are detected and cleared outside the model, and we assume that remote frames are filtered at some layer beneath the protocol to remove APS messages that do not appear in three consecutive APS frames (see Req. R5-80 of the Bellcore standard).

The core of the APS protocol is a function that computes switch outputs from switch inputs. The inputs are the technician command, the line conditions, and the APS message from the remote switch. The outputs are the selector setting, which determines the line to be used for reading, and the APS message to be sent to the remote switch. So for the protocol, the switch interface is as in Figure 3. The technician command is a *request* of the form (request type, channel identifier). A *request type* is a literal. The most important types are 'forced switch' (FS), 'signal failed' (SF), 'signal degraded' (SD), 'reverse request' (RR), and 'no request' (NR). Each request has a priority, which is most cases is determined by its type. A *channel identifier* is either 'null' (0), indicating that frames should be read from the working line, or 'working' (1), indicating that frames should be read from the protection line. The line monitor input is a pair of *line conditions*, where a line condition can be 'clear', 'degraded', or 'failed'. The selector setting is a channel identifier. Finally, an APS message has the form (request, channel id), where the *channel id* is the value that was received in the most recent request from the remote switch.

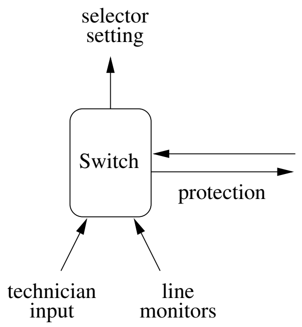

Figure 3. The Interface of an APS Switch

We now sketch how the selector setting and the APS message for the remote switch are computed. First we consider the selector setting. If the first channel of a switch's message matches the second channel of a switch's remote message, then this is the selected channel. Otherwise the null channel is the selected channel.

Computing the APS message for the remote switch is a two step process. In the first step a local request of the switch is computed according to the technician and line monitor input. If the technician has made a request or if the line conditions are not clear, then the local request is the higher priority of these; otherwise the local request is of type 'no request' (this type is simply the lowest priority value). In the second step the local request is compared with the request from the remote switch (see Fig. 2). Roughly, if the remote request has higher priority than the local request, then a request of type 'reverse request' is used in the message to the remote switch. If thef remote request has lower priority, or if the remote request is of type 'reverse request', then the local request is used. When a request of type 'reverse request' is made, the channel id of the request is the channel id found in the remote request.

3. Analysis Goals

To motivate our analysis it is useful to review the scope of the APS standard. The standard is intended to ensure that a SONET system built from a pair of switches will function properly, even if the switches are built by different manufacturers. To do so the standard specifies the behavior of a single switch, including its electrical interface, timing details, and message formats. However, the standard does not specify what it means for a system built from a pair of switches to function properly. So although the standard was designed to guarantee proper system behavior, it does not reveal the behavior that is guaranteed. Our analysis concerns both what the standard explicitly specifies about a single switch and what it implicitly specifies about a pair of connected switches.

Our first analysis question is whether the APS switch of the 5ESS conforms to the standard. One way of defining conformance, adopted by the ISO/ITU-T working group on Formal Methods in Conformance Testing [6], is to say that an implementation conforms to a specification if the implementation and specification represent semantic models that belong to an implementation relation. We call this notion of conformance *primary conformance*, or just *conformance*. For example, the semantic models could be finite-state machines, and the implementation relation could be trace equivalence. Then an implementation would conform to a specification if the sequences of actions that could be performed by the implementation are exactly those that could be performed by the specification.

Another way to define conformance is to say that an implementation conforms to a specification if it does its job just as well as an primary conforming implementation. More precisely, we require that an implementation interoperates properly with every implementation that conforms to the specification in the primary sense. We call this notion of conformance *coordination conformance*. An implementation can satisfy coordination conformance even if it does not satisfy primary conformance. However, the opposite is also true. The specification may be too weak, so that some implementations satisfying primary conformance do not interoperate properly.

Our second analysis question is whether any two conforming APS switches interoperate properly. We say that a specification of a system component is *valid* if any components that conform in the primary sense to the specification will, when connected, yield a system that interoperates properly. Like coordination conformance, validity depends on a notion of proper interoperation, and hence only makes sense for specifications of system components. Note that if a standard is valid, then every implementation that conforms to the specification in the primary sense also conforms to it in the coordination sense.

4. Validation

To establish the validity of the APS standard we need to know what 'proper interoperation' and 'primary conformance' mean for APS. These notions are not defined in the standard, so we were forced to guess what they should mean by studying the standard and related documents (e.g., [3]). Two main aspects of proper interoperation are captured in the following properties:

Agree A pair of switches should select the same channel soon after the line conditions settle.

Match Channel mismatches should cease soon after the line conditions settle.

For $\mathbf{Agree}$, 'soon' is roughly 50 ms; for $\mathbf{Match}$, it is about half a millisecond. The phrase 'line conditions settle' means that the status reported by the line monitors fails to change and that no new technician inputs are received.

To understand primary conformance we had to come up with semantic model of the specification and an implementation relation. A problem in defining the semantic model is that the standard does not say how a switch should interact with its environment. In other words, it does not specify when inputs are collected, when the outputs are computed, or when the outputs are emitted. Since SONET has a fixed frame rate, we assume that a switch performs a fixed sequence of actions whenever a frame is received from the remote switch. We assume that the switch first exchanges messages with the remote switch, then reads the technician input and line conditions, and then outputs the selector setting.

For our implementation relation we chose observation equivalence [18], a kind of bisimulation equivalence. This decision is explained at the end of this section. Since our switch specification is nondeterministic, observation equivalence here coincides with weak trace equivalence [18]. In other words, two switches will be observation equivalent just if they have the same traces. Consequently, a switch that performs input and output just as our specification does, and that computes its outputs as our specification does, will be observation equivalent to our switch specification.

Our next step was to formalize the switch specification and properties $\mathbf{Agree}$ and $\mathbf{Match}$. In formalizing the switch specification we first described it in pseudocode. From the pseudocode we derived a specification of a switch in the value-passing form of the process algebra CCS [18]. Process Switch_A, which specifies a switch identified by A, has actions $\overline{\texttt{rreq}}_A$, $\texttt{rreq}_B$, $\texttt{lreq}_A$, $\overline{\texttt{sel}}_A$, and $\overline{\texttt{match}}_A$. Action $\overline{\texttt{rreq}}_A$ represents the output of a request from switch A to the remote switch. Action $\texttt{rreq}_B$ represents the input of a request by switch A from the remote switch. Both $\overline{\texttt{rreq}}_A$ and $\texttt{rreq}_B$ are parametrized actions with requests as their parameter values. Action $\texttt{lreq}_A$ represents the input of the highest-priority local request, which may have originated from the line monitor or the technician. It is a parameterized action with a request as its value. Action $\overline{\texttt{sel}}_A$ represent the output of the selector setting by switch A. It is a parametrized action with value either p or w. Finally, action $\overline{\texttt{match}}_A$ represents an output showing whether the channel id in the remote message matches the channel id in the local message. It is a parametrized action with a boolean value.

Our first APS model had so many states that it could not be automatically be analyzed with the Concurrency Workbench on the computers available to us. To solve this problem we simplified the model. First, we removed the part of the model that captured the calculation of the highest priority local request from the line monitor and technician inputs. The simplification is justified in part by the observation that the complex parts of the standard are still captured in our switch specification. Second, we eliminated some request types from our model. In particular, we model none of the request types related to technician inputs. Finally, we removed the part of the model that captured the SONET lines between the switches. Our APS model instead allows the switches to communicate directly.

Our complete CCS model of APS has three processes: one for each switch, and one that models the switch inputs. Figure 4 shows the CCS flow diagram of the model. Process Switch_A and Switch_B model the switches. Process *Input* abstractly models the technician and the condition of the protection line. Actions $\overline{\texttt{req}}_A$ and req_B represent the highest-priority local requests for switches A and B. Action `change` models a change in the value of a highest-priority local request for either switch.

The properties $\mathbf{Match}$ and $\mathbf{Agree}$ were described as temporal logic formulas. The logic we use is the modal mu-calculus [17, 20], a propositional modal logic extended with fixed-point operators. We will not present the formulas here because there is not space to explain the logic. However, we can paraphrase the formulas. Informally, formula $\mathbf{Match}_s$

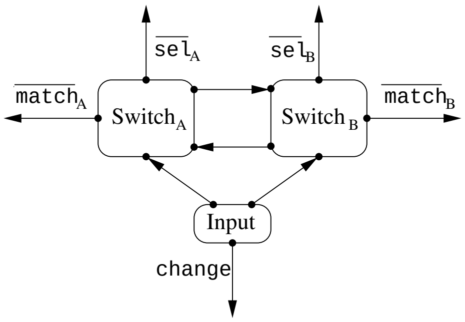

Figure 4. Flow Diagram of the Basic Model

holds of a process unless some trace of the process has a suffix along which no change of input occurs but mismatches continue to occur at switch s indefinitely. Formula $\mathbf{Agree}$ holds of a process unless some path of the process has a suffix along which no change of input occurs but both the working and protection channels are selected indefinitely. Notice that our formalization of the properties do not capture the real-time aspects mentioned in Section 3.

We analyzed our CCS model with the Concurrency Workbench of North Carolina (CWB-NC) [9, 8]. The tool shows that our CCS model has about 19,000 states and 160,000 transitions, and that it does satisfy formulas $\mathbf{Agree}$ and $\mathbf{Match}$. Verification of each property takes about 20 minutes on a Sun Ultra Enterprise E3000 with a gigabyte of memory and 4 UltraSPARC processors.

These analysis results tell us about our switch specification, not about switch implementations. However, from results about CCS and the modal mu-calculus [21] we know that $\mathbf{Agree}$ and $\mathbf{Match}$ will also hold if the two copies of the switch specification are replaced by switch implementations. Briefly, this is so because we have chosen observation equivalence as our implementation relation, and observation equivalence is preserved by all CCS operators except choice. Furthermore, two observation equivalent processes share all the same formulas of the weak form of the modal mu-calculus, which is the form used to formalize properties $\mathbf{Agree}$ and $\mathbf{Match}$.

We also analyzed a model in which protection line failures occur. We model this case by corrupting protection line messages when the line monitor shows the protection line to have failed. This case is important because APS messages are carried in SONET frames on the protection line. Furthermore, the channel to select in the presence of a channel mismatch is critical here, yet the standard does not specify it (it should be specified in requirement R5-76). In our model the protection channel is selected in the presence of a channel mismatch. Using the CWB-NC we found that formula $\mathbf{Agree}$ holds of our revised model but $\mathbf{Match}$ does not. Mismatches can continue to occur because a switch with a low-priority local request may never see the high-priority request of the other switch.

5. Conformance Analysis

Our second major goal was to check that the 5ESS implementation of APS conformed to the APS standard. As described above in Section 3, a valid APS standard and a means of checking that the 5ESS implementation satisfies the primary-conformance property would assure that it would interoperate properly with any other primary-conformant implementation. However, such a check would assure proper interoperation only if we knew for certain that the APS standard satisfies the validity property. Since we could not completely establish this property for APS, we also defined the notion of coordination conformance. Primary conformance requires the demonstration of an implementation relation between a formal specification of APS and an implementation, while coordination conformance focuses on the ability of an implementation to interoperate correctly with any other primary-conformant one. As a means of detecting as many errors as possible in the 5ESS implementation, we decided to perform separate checks for primary and coordination conformance. The next two subsections describe our primary- and coordination-conformance analyses in more detail, along with the results of each analysis.

5.1 Coordination Conformance

The properties $\mathbf{Agree}$ and $\mathbf{Match}$ specified in the last section capture the essence of what it means for two interconnected APS switches to interoperate properly. To verify that the 5ESS implementation of APS satisfies the coordination-conformance property, we must therefore check that $\mathbf{Agree}$ and $\mathbf{Match}$ hold of a system composed of it and a primary conformant APS implementation. We used the VeriSoft tool [12] for this purpose. Like the Concurrency Workbench, VeriSoft performs an automated analysis of a network of communicating processes. However, VeriSoft allows the code of a concurrent system to be analyzed directly, without first describing the system's behavior in a modelling notation like CCS. To use VeriSoft to check that the 5ESS implementation satisfies the coordination-conformance property, we needed a second, primary-conformant APS implementation in C. Toward this end, we translated the pseudocode described above, which was the basis for our CCS specification of APS, into the C programming language. Because we have defined the pseudocode to be a precise statement of the APS protocol, we assume that our C implementation of it is primary-conformant. Of course, we have no way of proving this fact, since the C code may contain errors and we may have misinterpreted the APS standard. Thus, a useful additional benefit of this analysis is to detect cases in which we and the 5ESS implementors of APS interpreted the standard differently, thereby identifying ambiguities.

For both of our conformance analyses, it was necessary to isolate the portions of the 5ESS implementation concerning the APS protocol logic so that they could execute independently of the 5ESS execution environment. The 5ESS APS protocol is implemented in a combination of C code and the Virtual Finite State Machine (VFSM) notation ([22, 11]). VFSM allows the top-level control structure of a module to be modeled as an extended finite state machine. A translator generates C code directly from the VFSM specification. Large applications are often decomposed into several VFSMs that communicate by asynchronous message passing. The VFSM APS implementation consists of 14 VFSMs. Four of these VFSMs are concerned with implementing the APS protocol; the remaining ten handle such functions as interfacing with the working and protection lines to detect signal degradations and failures. Both of the analyses described below were carried out using the four VFSMs that implement the APS protocol.

In VeriSoft, each process in the system is represented as a Unix process, and processes interact via Unix communication primitives, such as semaphores and message passing. VeriSoft systematically searches the state space of a network of communicating Unix processes. It checks for generic concurrency problems like deadlock and livelock, and can also detect violations of user-specified assertions inserted into the C code of each process. To construct a concurrent C program of two connected APS switches that could run under the control of VeriSoft, we needed to model the protection line in C so that the switches could exchange APS messages. We modelled each direction of the protection line as a queue. In an APS system each switch sends a message to the remote switch at the SONET frame rate. However, our model has no clocks to ensure that the switches run at about the same rate. We keep the switches of our model in near synchrony by making the size of the queue between them equal to one. We modelled the line monitor and technician inputs with a pseudo-random function that generates possible sequences of line-condition changes and technician commands.

The property $\mathbf{Match}$ states that channel mismatches in the APS messages should cease once the system quiesces after processing technician commands and line-condition changes. To check this property in VeriSoft, we allowed each switch to generate a finite number of technician commands and line-condition changes. We added a VeriSoft assertion to each switch which states that once this number of commands and changes has occurred and the system quiesces, the locally requested channel (the first channel of the local message) should equal the remotely requested channel (the first channel of the remote message). Unfortunately, we were unable to check the property $\mathbf{Agree}$ in this way: in the current version of VeriSoft, one cannot specify assertions that reference variables of more than one process, and checking $\mathbf{Agree}$ would require access to the selected channels of both the 5ESS and C implementations.

VeriSoft explored several million states of the concurrent APS system described above, and did not find any violations of the property $\mathbf{Match}$.

5.2 Primary Conformance

Our next step was to establish that the 5ESS implementation satisfies the primary-conformance property. The results of Section 4 indicate that if an implementation is observation equivalent to our CCS model of APS, a system composed of that implementation and another primary-conformant one will satisfy the proven properties. However, no automated tools exist for showing that a C program is observation equivalent to a CCS process. Section 4 also notes that, for our APS specification, observation equivalence reduces to functional equivalence, which can be demonstrated by showing that the 5ESS implementation computes the same function from inputs to outputs as a specification of the APS protocol. We therefore investigated how we might use the C implementation we created for the coordination-conformance analysis to perform a primary-conformance check. The essential function of an APS implementation is to compute a message to send to the remote APS switch that is a function of a number of inputs. As a way of checking that the 5ESS implementation "implements" a primary-conformant APS switch as embodied in our C implementation, we decided to see if the two implementations computed the same APS message for each possible combination of inputs. No general-purpose tools were available for this purpose, so we wrote a special-purpose program that generated all input combinations; for each one, it executed both implementations with that combination as input and compared the APS messages computed as output.

Constructing this special-purpose program posed a number of problems. The inputs to our C program were easily accessible as shared variables and parameters to C functions, so it was not difficult to set them to a particular combination of values and execute the program. The corresponding inputs to the 5ESS implementation were represented in the C code generated by the VFSM translator and could not be easily accessed. We therefore represented each input as a message that we placed on the appropriate VFSM queue and executed the VFSMs until all these messages were processed, thereby setting the VFSMs to the desired internal configuration. Another difficulty was that the computation in the 5ESS implementation of the selected channel and the second channel number in each APS message are not carried out by the VFSM implementation, so we had to write some extra C code to compute them in our program. As a result, we could only compare the APS requests (i.e., the request type and first channel number) in each APS message produced by the two implementations. Producing the special-purpose program thus required a detailed understanding of both implementations and the writing of considerable code to produce an environment in which both could be executed so their outputs could be compared.

When we ran the program, the outputs of the 5ESS implementation differed from those of our C implementation on 552 of the 15,750 possible input combinations, or about 3.5% (we also found a bug in our C implementation that was causing it to ignore the clearing of technician commands). In examining the cases where the implementations disagreed, we found that each was an instance of one of three root causes, all of which were due to differences in how we and the 5ESS developers interpreted the standard. For illustration, we now briefly describe one of these root causes. For brevity in what follows, we refer to the protection and working channels with the numbers 0 and 1, respectively, and denote request tuples of the form (FS, 0) as simply FS0.

The difference we will describe between the two implementations came about because the APS standard says little about the interface through which technician commands are conveyed to the APS protocol logic. The standard states that a technician command can be accepted only if it has strictly higher priority than the current highest-priority request at a switch. Thus, for example, if FS0 is currently being sent and the technician issues an FS1 command, the latter should be rejected, since its priority is equal to that of the FS1. However, the technician can first issue a 'clear' command to remove the FS0, and then the FS1 would be accepted. As a convenience to the technician, the 5ESS implementation contains a feature known as *smart switching* that allows the technician to issue the FS1 in the above example without first clearing the FS0. Since our implementation adheres to the standard and does not allow smart switching, it disagreed with the 5ESS implementation whenever an FS command was issued while another FS for a different channel was being sent. Interestingly, the smart-switching feature created problems when the 5ESS implementation was tested against an implementation of APS produced by another Lucent development organization that did not support smart switching: the latter refused to allow the technician command, causing a channel mismatch. As a result, the 5ESS developers modified their implementation so that, in the above example, the technician could still issue FS1 while FS0 was being sent, but the FS0 would be cleared at the APS-protocol level before the FS1 is sent. Note that while our C implementation does not allow smart switching locally, it will permit smart switching by a remote switch connected to it; this is why our VeriSoft analysis did not detect a channel mismatch.

6. Summary and Suggestions

Our study of APS is unusual in that we looked at both the validity of the standard and whether a particular APS implementation conforms to the standard. To do this we used various tools, including a model checker, a state space exploration tool for reactive C programs, and a analysis tool built specially for our analysis. Besides identifying problems with the standard, we also defined a new notion of conformance that we call 'coordination conformance'. A system that conforms in this sense is guaranteed to interoperate properly with every system that conforms in the usual sense.

The most serious problems we faced in our analysis stemmed from the standard, which does not clearly specify what a switch should do, what it means to implement a switch, or what is guaranteed about about a switch built from switch implementations that satisfy the standard. For our analysis we could only decide these issues for ourselves.

Other work on the analysis of protocols also suffers from the failure of standards to define proper interoperation and primary conformance. In [10], Ferguson describes a validity analysis of a data link layer protocol for digital cellular mobile radio. Ferguson takes proper interoperation to mean that: no messages are lost, duplicated, or reordered; termination only occurs as intended; and no 'non-progress' cycles exist. Data link protocols are common and, unlike APS, the properties expected of them are usually well understood. However, the analysis does not address the issue of conformance. Therefore the analysis does not really show that the protocol is valid in our sense of the word, since Ferguson, and presumably also the standard, does not say what it means for an implementation to satisfy the standard.

In [19], Sighireanu and Mateescu describe a validity analysis of the link layer protocol of the IEEE-1394 serial bus. Again, the standard apparently does not define proper interoperation, but unlike [10], here proper interoperation is understood as the conjunction of five complex properties, which were formalized in temporal logic. The complexity of these properties suggests that they may not accurately reflect what the designers of the protocol understood as proper interoperation. Again, conformance is not addressed.

The holes and ambiguities of the standard could have been avoided if it were written in a well known programming language such as C. The C language has been used in the specification of other standards (e.g. Scalable Coherent Interface (SCI)[14]). C-like formalisms exist for concurrent systems (e.g., UNITY [7] and Promela [13]). Similarly, Pascal is used in the IEEE 802.3 CSMA/CD protocol standard for LANs [15]. Our 5ESS development colleagues found our C description of the protocol easy to understand and superior to that which appeared in the APS standard. The lack of abstraction mechanisms are an apparent drawback of C, but abstraction can be introduced through an implementation relation.

A more ambitious suggestion is that protocol standards explicitly define 'primary conformance' and 'proper interoperation'. Doing so would allow independent conformation of the validity of a standard. Indeed, the validity of the standard could be defended in a separate report by a standards organization. Defining these concepts in the standard would additionally make it possible for manufacturers to know what they could guarantee about their equipment. In light of the difficulty we faced in expressing proper interoperation for APS in temporal logic, it is unlikely that temporal logic could be used now for this purpose in standards. However, in the short term the notions could be defined in English.

A final problem we will discuss here was how to modify the VFSM implementation of APS to use the VeriSoft send and receive primitives. The difficult part of the modification was simply to locate the points in the VFSM implementation that represent the send and receive operations of a switch. Languages like C do not have syntactic constructs for I/O ports. Instead they have operating system supported I/O functions that can be extended by library functions or by application program code to provide higher-level logical I/O. If a programming language did have a construct for identifying logical I/O ports then it would be easy to find the fragment of the program needed for testing purposes.

7. Acknowledgements

Mark Ardis, Michael Benedikt, Bill Bielawski, Peter Fales, Patrice Godefroid, and Lalita Jagadeesan helped us to understand the APS protocol and made suggestions about its analysis. Bill Bielawski and Peter Fales discussed alternative interpretations of the standard and their possible problems.

References

[1] Digital hierarchy – Optical interface rates and formats specifications (SONET), ANSI T1.105-1991, 1991. American National Standard for Telecommunications.

[2] J. Bellamy. *Digital Telephony*. John Wiley & Sons, second edition, 1991.

[3] SR-NWT-001756, Automatic protection switching for SONET, issue 1, October 1990. Bellcore.

[4] GR-253-CORE, Synchronous optical network (SONET) transport systems: Common generic criteria, issue 1, December 1994. Bellcore.

[5] GR-253-CORE, Synchronous optical network (SONET) transport systems: Common generic criteria, issue 2, December 1994. Bellcore.

[6] A. R. Cavalli, J. P. Favreau, and M. Phalippou. Standardization of formal methods in conformance testing of communication protocols. *Computer Networks and ISDN Systems*, 29(1):3–19, 1996.

[7] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[8] R. Cleaveland. The NCSU concurrency workbench. In *Proceedings of CAV '96, LNCS 1102*, pages 394–397, 1996.

[9] R. Cleaveland and S. Sims. The NCSU concurrency workbench. In R. Alur and T. Henzinger, editors, *Proceedings of CAV '96*, 1996.

[10] M. J. Ferguson. Formalization and validation of the radio link protocol (RLP1). *Computer Networks and ISDN Systems*, 29(3):357–372, 1996.

[11] A. R. Flora-Holmquist, E. Morton, J. D. O'Grady, and M. G. Staskauskas. The virtual finite state machine design and implementation paradigm. *Bell Labs Technical Journal*, 2(1):96–113, Winter 1997.

[12] P. Godefroid. Model Checking for Programming Languages using VeriSoft. In *Proceedings of the 24th ACM Symposium on Principles of Programming Languages*, Paris, January 1997.

[13] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[14] IEEE. Standard for scalable coherent interface. ANSI/IEEE Standard 1596-1992, 1992.

[15] IEEE. ANSI/IEEE Std 802.3, CSMA/CD access method and physical layer specifications, 1996.

[16] Recommendation G.781 – Structure of recommendations on equipment for the synchronous digital hierarchy (SDH), 1993. International Telecommunications Union.

[17] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[18] R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[19] M. Sighireanu and R. Mateescu. Validation of the link layer protocol of the IEEE-1394 serial bus ("FireWire"): an experiment with E-LOTOS. Technical Report 3172, Institut National de Recherche en Informatique et en Automatique, 1997.

[20] C. Stirling. Modal and temporal logics for processes. Technical Report ECS-LFCS-92-221, Laboratory for Foundations of Computer Science, University of Edinburgh, 1992.

[21] C. Stirling. Modal and temporal logics for processes. Notes for Summer School in Logic Methods in Concurrency, Computer Science Department, Åarhus University, Denmark, 1993.

[22] F. Wagner. VFSM executable specification. In *Proc. IEEE International Conference on Computer System and Software Engineering*, pages 226–231, The Hague, Netherlands, 1992.