Danny

IEN-79

Tomas Lozano-Perez (TLP@CCA)
Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139
January 1979

A Protocol Test Facility: Request for Comment

"Introduction"

This short paper is a request for comments on a new approach to testing and debugging protocol implementations. Our goal is to design a tool to make testing and debugging protocol implementations easier. The concept is based on providing a central, network-wide facility to test protocol implementations.

Users of this facility would have available to them a whole range of programs to test particular protocol implementations. For example, one program might engage in an NVP vocoder negotiation; another might test the connection procedure of a TCP implementation. Users could also write their own test programs, either as variations of existing ones, or new ones aimed at exercising one particular aspect of their implementation. The tests would be run at a central network site and the resulting communication trace as well as any error conditions and diagnostics would be returned to the user or be available for later study at the central site.

The tests would be specified in a high level language designed specifically for this purpose. The test programs can simulate "correct" and "incorrect" protocol behavior on the part of the tester. The high-level language not only simplifies the task of writing the tests but also serve as an unambiguous definition of the tester's behavior and of correct behavior in those circumstances.

An important aspect of our testing concept is that a network test facility will provide a single network-wide performance standard for protocol implementations. This contrasts to the current situation where protocol testing is most often done by independent pairs of hosts. This process tends to produce "dialects" of the protocol. Due to the tendency of network "cliques" to develop, i.e. the group of hosts that tested together mainly communicate with each other, incompatible implementations are slow to be found. When they are found, further implementation and testing is required. This standardization cycle can take a long time to converge. We believe that a central protocol test facility will speed the initial convergence while also being a powerful tool for maintenance.

"Current Protocol Testing Methods"

In current practice, testing a network protocol consists of attempting to use the fledgling protocol implementation to communicate with an existing implementation of the protocol. During the early phases of development this might be the other half (i.e. receive vs send) of the same
implementation. During later development, testing amounts to attempting normal communication with implementations on other hosts. Implementors rely on this type of testing because it is not economical for users to develop their own testing program.

This kind of program testing is unsatisfactory for a variety of reasons, having to do with the lack of control over the distant host's implementation:

(1) The kind of tests that can be run is limited and depends on the particular set of implementation decisions the other implementor made, e.g. retransmission strategies, timeouts, etc.

(2) Test scenarios are limited to those involving correct protocol operation on the part of the distant host (or to the particular form of incorrect behavior to which the foreign host considers correct). The tester cannot force the other host to selectively violate the protocol or to crash on cue.

(3) During the early phases of development, it might be difficult to find a host (and a person) willing to cooperate in the testing.

(4) In general, it is difficult to limit tests to specific portions of the protocol, e.g. initial connection or reset procedures.

After running the tests, the analysis is still a difficult task; one problem is that it is hard to tell what went wrong during a particular test. The are several reasons for this:

(1) One usually does not know exactly what messages were received by the foreign host.

(2) Since the implementation is not completely specified by the protocol constraints, one cannot tell what the other program did.

(3) Most protocol error messages are useless as a diagnostic since they are defined for all possible implementations and thus tend to be very general.

These problems are clearly the result of carrying out tests in a situation not intended for testing; normal protocol implementations are not test programs. All of these factors conspire to make protocol testing and debugging extremely painful. The process is further complicated by the fact that, currently, protocols are specified in a natural language and are subject to all of its ambiguities. Many (if not most) failures during initial testing of a protocol implementation are due to conflicting interpretations of the protocol specification. The result is that most implementations are inadequately tested when they first become "operational".

"The Network Test Facility"

The previous section brought up four kinds of deficiencies of current protocol testing techniques:

(1) Lack of a unique testing "reference";

(2) Lack of control by the tester;

(3) Unpredictable behavior on the part of the foreign host;

(4) Problems with unpredictable network behavior, e.g. lost or garbled messages.

Most of these problems can be significantly reduced by means of a network facility designed with protocol testing in mind. The key idea is to provide a large set of test programs written in a specially designed high level language. The user can then specify parameters to these tests and so cover a nearly complete range of test scenarios. The user can also write tests if he so desires; although, this is not envisioned as the usual mode of use. The tests would amount to a functional specification of the protocol. They would be written early on, as the protocol was being specified.

Test specifications would be executed at one or more network sites causing a controlled and repeatable interaction with the implementation under test. This facility could also record the interactions and provide diagnostic messages on the reasons for failure. The use of such a facility does not remove the problems associated with unpredictable network behavior, but it reduces the interaction of such effect with other uncertainties.

Two basic questions must be discussed about such a facility:

(1) How are the tests specified;

(2) How are the tests used.

These questions will be addressed in the next sections.

"Specifying Protocol Tests"

What precisely is a protocol test?

A protocol test is a predictable sequence of messages between two (or more) processes, which follows the syntactic conventions of the protocol. The important thing is to be able to tell whether the behavior of one of the processes satisfies the protocol definition. If not, the source of the violation should be pinpointed.

Specifying a test for a protocol implementation, is akin to specifying a protocol. It is not as simple as providing a list of messages to be sent, since correct protocol behavior normally requires reacting to messages received. For this reason, tests must be specified as programs.

Protocol specifications involve a syntactic and a semantic component. The syntax of the protocol specifies what constitutes a legal message, i.e. bit patterns, checksums, etc. There are two components to the semantics: (1) effects on the communication state, i.e. receiving or sending a message affects what other messages are legal; and (2) side-effects, e.g. the delivery of data, playback of voice, etc. Protocol tests must satisfy the syntactic conventions of the protocol and be able to detect violations to it. Similarly, the test must be able to implement the restrictions on communication state specified by the protocol, e.g. connection establishment or flow control. In addition, violations to these specifications must be detected. It is sometimes desirable to test the behavior of the implementation to incorrect input so a test should be able to produce incorrect behavior sequences. It is very hard for the tester to determine whether a communication is having the desired side-effects at the foreign host. This checking must be done by the user based on the test specification.

Test specifications, like the protocol specifications they resemble, focus on the changes in communication state. The syntax is assumed to apply to all transmissions, i.e. is treated as a SEND and RECEIVE subroutine and the side-effects are not described explicitely. This type of specification works better for host-host protocols than for high-level protocols where the "side-effect semantics" are more difficult.

"PROTEST: A PROtocol TESTing language"

The test specification technique we are proposing is modelled on Augmented Transition Networks (ATN), a grammar specification technique described in [Woods] for application to parsing natural language sentences. It is a state based technique, like many proposed protocol specificatin techniques. The ATN extends the finite state machine formalism to general programming language power by means of the following augmentations:

(1) General tests on the arcs (not just equality test on input).

(2) General actions on the arcs, e.g. setting variables, calling subroutines, output functions, etc.

(3) PUSH/POP actions which provide the ability to use an ATN as a (recursive) subroutine. This mechanism is very important since it allows hierarchical definition of the state diagram as well as capturing similarities between state sequences.

Let us consider an ATN model of a very simple data transmission protocol (originally defined in [Stenning] as a PASCAL program). The protocol has two components, a transmitter and a receiver, both are shown below in table format. Each one can be used to test the correct implementation of the other.

TRANSMITTER

STATE	TEST	NEXT	ACTION
A	--	B	LU := ISN + 1 HS := ISN
B	HS-LU < TW-1 & EXIST MSG(HS+1)	B	HS := HS+1 SEND MSG(HS)
	TIMEOUT(I)	B	SEND MSG(I)
	RCV ACK(I)	C	
	LU > HS	-	POP

C	HS >= I >= LU	B	LU := I+1	
	I < LU	B	IGNORE ACK(I)	

RECEIVER

STATE	TEST	NEXT	ACTION
D	--	E	RECEIVED(*) := 0 NR := ISN
E	RCV MSG(I)	F	RECEIVED(I) := RECEIVED(I)+1
F	RECEIVED(NR) >= 1	E	NR := NR + 1 SEND ACK(NR-1)
	--	E	SEND ACK(NR-1)

The operation of the protocol depends on the following variables:

ISN - The initial sequence number, presumably determined during the connection extablishment phase.

LU - is the Lowest Unacknowledged sequence number of any of the messages sent.

HS - is the Highest Sequence number sent.

TW - is the Transmit Window size, i.e. the number of unacknowledged messages allowed.

Each ACK(i) acknowledges receipt of all messages with sequence number less than or equal to i. The protocol simply sends a message whenever the number of unacknowledged messages (HS - LU) is less than the transmit window size. Whenever a message is sent, HS is incremented and whenever an ACK is received, LU is set to 1 plus the ACKnowledged sequence number. When all messages are acknowledged and no more

messages remain to be sent, a POP action is executed which returns to the caller.

The receiver keeps an (possibly infinite) array indicating which messages have been received. NR indicates the next sequence number expected in the receiver, i.e. the lowest unacknowledged sequence number. When a message with sequence number i is received, it is marked as such in the RECEIVED array. NR is then incremented as long as RECEIVED(NR) is > 0. After this cycle NR is again the lowest unacknowledged sequence number, so ACK(NR-1) is sent.

Notice that these two ATN's only specify the cycling of the communication state. All the syntactic constraints are embodied in the SEND and RECEIVE procedures. It is these procedures which actually build network messages, carry out the checksum computation, etc.

The two networks above can be used to test implementations of the protocol. They in turn can be used as part of other networks, e.g. a test sequence that involves alternating the transmitter and receiver role. A slight variation of the receiver ATN also allows testing the retransmission mechanism of the transmitter. The receiver currently tests if RECEIVED(NR) >= 1 before acknowledgeing any new messages. We can change the 1 to REPEAT, some arbitrary number. This forces the transmitter to SEND each message at least REPEAT times (maybe more, since messages may be lost in transit). This illustrates the control possible with this kind of testing, which is lacking when another implementation is used for testing.

"Using PROTEST to test protocol implementations"

The PROTEST system would run on a small stand alone machine, and be available around the clock. Sessions would be arranged by a supervisor/logger process, whose job is scheduling times and resources and supervising the data base of specification files. We expect that normal use of the facility will involve executing a pre-existing test, but the user can also specify a test by simply shipping a file with the definition of an ATN to the monitor process. Tests can be combined or iterated by specifying simple ATN's that use other tests as subroutines (via PUSH actions).

The key component of the PROTEST system is an interpreter. This program executes tests written in the PROTEST language. The program is driven by input from a message I/O module. This module implements the SEND and RECEIVE actions. It takes messages from the IMP interface and arranges them into a structure specified by a DATA FORMAT component of the test specification. This allows symbolic access to the messages. The

interpreter produces messages which are converted by the I/O routine into legal network messages (by adding the header, etc.). Running a test generates a trace file of all the messages recieved and sent during a test session and the associated state of the ATN being run. Any error and diagnostic information produced by the system is included in a separate file. The error messages indicate the state where failure occurs; this defines the range of correct responses in the situation.

"Conclusions"

We believe the kind of facility we describe here to be highly desirable in an environment where many different implementations of a protocol are being developed and maintained. It should help guarantee certain minimum standards of compatibility among the implementations without restricting the implementation details. The test sequences can also serve as a functional definition of correct protocol behavior.