To be published in the proceedings of the 1996 International Symposium on Software Testing and Analysis, San Diego, CA, January 1996.

An Approach to Verification and Validation of a Reliable Multicasting Protocol

John R. Callahan and Todd L. Montgomery
{callahan,tmont}@cerc.wvu.edu
NASA Software IV&V Facility
100 University Drive
Fairmont, WV 26554
304-367-8235, 304-367-8211 (fax)

Abstract: This paper describes the process of implementing a complex communications protocol that provides reliable delivery of data in multicast-capable, packet-switching telecommunication networks. The protocol, called the Reliable Multicasting Protocol (RMP), was developed incrementally using a combination of formal and informal techniques in an attempt to ensure the correctness of its implementation. Our development process involved three concurrent activities: (1) the initial construction and incremental enhancement of a formal state model of the protocol machine; (2) the initial coding and incremental enhancement of the implementation; and (3) model-based testing of iterative implementations of the protocol. These activities were carried out by two separate teams: a design team and a V&V team. The design team built the first version of RMP with limited functionality to handle only nominal requirements of data delivery. This initial version did not handle off-nominal cases such as network partitions or site failures. Meanwhile, the V&V team concurrently developed a formal model of the requirements using a variant of SCR-based state tables. Based on these requirements tables, the V&V team developed test cases to exercise the implementation. In a series of iterative steps, the design team added new functionality to the implementation while the V&V team kept the state model in fidelity with the implementation. This was done by generating test cases based on suspected errant or off-nominal behaviors predicted by the current model. If the execution of a test in the model and implementation agreed, then the test either found a potential problem or verified a required behavior. However, if the execution of a test was different in the model and implementation, then the differences helped identify inconsistencies between the model and implementation. In either case, the dialogue between both teams drove the co-evolution of the model and implementation. We have found that this interactive, iterative approach to development allows software designers to focus on delivery of nominal functionality while the V&V team can focus on analysis of off-nominal cases. Testing serves as the vehicle for keeping the model and implementation in fidelity with each other. This paper describes (1) our experiences in developing our process model; and (2) three example problems found during the development of RMP. Although RMP has provided our research effort with a rich set of test cases, it also has practical applications within NASA. For example, RMP is being considered for use in the NASA EOSDIS project due to its significant performance benefits in applications that need to replicate large amounts of data to many network sites. The RMP source code and documentation are currently available on the WWW via http://research.ivv.nasa.gov/.

This work is supported by NASA Cooperative Agreement NCCW-0040 under supervision of the NASA Independent Software Verification and Validation (IV&V) Facility, Fairmont, WV.

1.0 Introduction

Much work has been done in the area of verifying that implementations of communication protocols conform to their specifications [1,2]. Conformance is usually verified through extensive testing of an implementation in which tests are derived directly from the protocol specification. If an implementation behaves in a manner predicted by the protocol specification, then the implementation is said to conform to the specification. If not, then an error exists in the implementation of the protocol. Although this method does not formally verify that a protocol specification and an implementation are consistent, it represents the state-of-the-practice in this domain of software development.

This paper describes our experiences while trying to formally specify and implement a complex communications protocol that provides reliable delivery of data in multicast-capable, packet-switching telecommunications networks. The protocol specification, called the Reliable Multicasting Protocol (RMP), was developed concurrently with its implementation. The implementation was developed incrementally using a combination of formal and informal techniques in an attempt to ensure the correctness of its implementation with respect to the evolving protocol specification. We found that many formal methods did not help us in the development of the protocol specification nor its implementation. We concluded that the best uses for formal methods in our situation was in the specification of the protocol requirements and the generation of tests derived from the specifications applied to prototype versions of the software during development.

One of the primary goals of our effort was to achieve high-fidelity between the specification and implementation during development. High-fidelity means that the specification model and implementation agree regarding the behavior of the protocol. We felt that if fidelity was not a primary concern, then there existed the strong possibility that the specification and the implementation would diverge in behavior. This would render analysis of any formal specification model irrelevant in the development and maintenance of the software since such analysis would offer little assurance that the actual code behaved in an identical manner.

Our development process involved two teams: a design team and a verification and validation (V&V) team. These two teams worked in an iterative, interactive fashion that allowed the design team to focus on nominal behaviors of the software while the V&V team examined off-nominal behaviors. The task of the design team was (1) to specify the protocol in terms of mode tables and (2) implement the protocol in C++ as specified by the mode tables. The task of the V&V team was to (1) analyze the consistency and completeness of the mode tables by analyzing "paths" through the mode tables and (2) generate tests from the mode tables for suspect conditions. Suspect conditions include those paths identified in the mode table model as being deadlock, livelock, or potential sources of unexpected behaviors. The V&V team used the requirements mode model to identify cases that were considered by the design team to be unusual or virtually impossible. In retrospect, these cases were the source of several errors in the specification and implementation of RMP.

We use the terms "verification and validation" in a different context from their typical usage because of our bipartite, prototyping development process. In our case, the term "verification" refers to activities that help in the identification of off-nominal behaviors of the software based on analysis of the specification model. We use the term "validation" to refer to activities that involve testing the implementation for properties based on potential problems revealed through verification analysis.

The protocol specification as expressed in the mode tables helped us organize and structure tests while developing implementation prototypes. Testing formed the dialogue by which the two teams communicated about the intended behavior of the protocol and its implementation. This paper relates our experiences in developing our approach and describes details of our model-based testing methods. We do not claim to have "formally verified and validated" the RMP specification and its implementation, but rather we have developed a strategy and process by which the evolution of RMP is enhanced by testing and verification. Our approach has been to study the problems that have occurred during development, testing, and operation of RMP. Through a post-mortem analysis of problems, we are trying to find methods that may have discovered problems earlier in the development lifecycle.

2.0 The Reliable Multicasting Protocol (RMP)

The Reliable Multicasting Protocol (RMP) [3] is based on an algorithm originally developed for reliable delivery of data in broadcast-capable, packet-switching networks [4]. The original algorithm, which we call the Token Ring Protocol (TRP), allows sites in a packet-switching network to establish a token ring for distributing responsibility for acknowledgments. A single token is passed from site to site around the ring and only the holder of the token (called the current token site) can acknowledge certain data packets. RMP has high-performance characteristics because acknowledgments themselves are multicast to all other token ring sites. When a site gets the token (i.e., it becomes the current token site), it multicasts an acknowledgment if and only if it has seen all data packets since the last acknowledgment it received. The token is passed in the multicast acknowledgment packet. The acknowledgment packet includes the source and sequence numbers of data packets it is acknowledging. This allows each site to detect if any packets are missing. A site will use negative acknowledgments to request retransmission of any missing packets. When all packets since the last acknowledgment received have been received by the current token site, then that site can multicast its acknowledgment and thus pass the token to the next site on the ring. When a token site sends an acknowledgment, it is assumed that all data packets since it last held the token have been received by all sites.

The sender of a packet assumes that all messages since it last had the token have been received by the other sites within a requested quality of service (QoS) level. A packet is marked delivered if and only if it satisfies its QoS level of delivery. The QoS level allows for resilience of the protocol in the presence of site failures and network partitions. In the case of failures, the token ring reforms itself around the failed site. In the presence of persistent failures, the application program using RMP must decide to degrade the QoS level or try again.

RMP differs from previous reliable broadcast protocols like TRP in that an acknowledgment packet may acknowledge an arbitrary number of data packets. Previous protocols specified that each data and acknowledgment packets have a one-to-one relationship. This dramatically

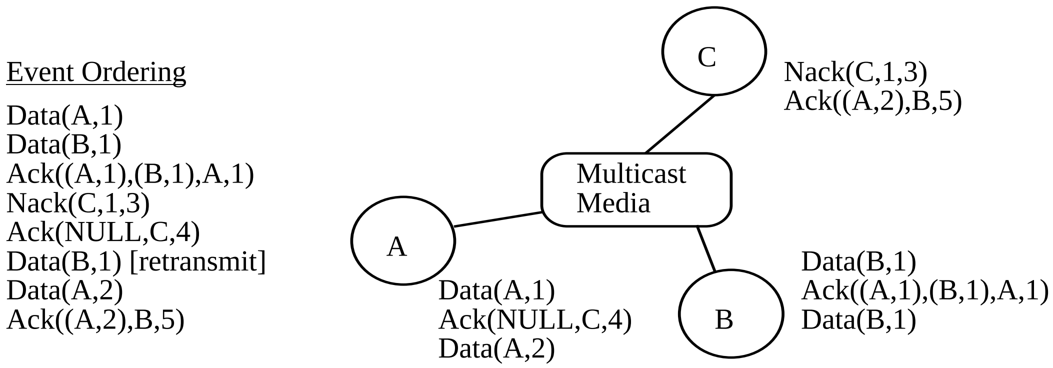

Figure 1: An example of an RMP token ring and events

improves throughput in networks with sporadic losses and allows an application to tradeoff performance and quality of service requirements.

Each site in a token ring maintains a data structure called an Ordering Queue (OrderingQ) in which acknowledgments and data packets are organized based on timestamps. An Ordering Queue is consistent if and only if there are no missing data packets for pending acknowledgments. A missing packet will appear as an empty slot in the OrderingQ that must be filled. When a site becomes the token site, all empty slots in the OrderingQ since the last acknowledgment received must be filled. For example, in Figure 1 we show 3 sites of a token ring and a global sequence of events. No site has complete knowledge of this sequence. It is only shown to illustrate a possible scenario. Next to each site is a list of the messages sent by that site. First, site A sends a data packet signified as Data(A,1) where the first parameter is the sending site and the second is the sequence number of the message. Sequence numbers are unique to individual sites. Second, site B sends a data packet (Data(B,1)). The initial token site is site B who then acknowledges both data packets and passes the token to site A. The Ack((A,1),(B,1),A,1) message contains a list of source identifiers and sequence numbers for two packets, followed by the next token site and the timestamp of the acknowledgment.

We assume that site C missed the data packet Data(B,1). Table 1 shows a snapshot of the OrderingQ data structure at site C after it receives the Ack((A,1),(B,1),A,1) message. Upon receiving this acknowledgment, site C realizes it has missed the Data(B,1) message that should fill the third slot of the OrderingQ. It knows this because the Data(B,1) packet is listed in the Ack message from B. Each slot in an OrderingQ corresponds to a timestamp whether explicit in the case of Ack messages or implicit in the case of Data packets. Site C will multicast a Nack message to request the data packet to fill the one slot in its OrderingQ at timestamp 3.

After a period during which no data packets are transmitted, Site A will time-out and subsequently send a multicast NULL Ack packet with timestamp 4. This passed the token to site C. Site B responds to the Nack by retransmitting the Data(B,1) message. The sequence number identifies this message uniquely to distinguish it from new messages. After the retransmission of Data(B,1), site A multicasts another data packet with sequence number 2 as Data(A,2). Since site

Packet	Timestamp	Token Pass or Data	Number of Packets
Ack	1	B -> A	2
Data	2	(A,1)	
Data	3	missing	

Table 1: Ordering Queue for Site C with empty slot

Packet	Timestamp	Token Pass or Data	Number of Packets
Ack	1	B -> A	2
Data	2	(A,1)	
Data	3	(B,1)	
Ack	4	A -> C	0
Ack	5	C -> B	1
Data	6	(A,2)	

Table 2: Final Ordering Queue for Site C

C's OrderingQ is consistent, it multicasts an acknowledgment of the Data(A,2) packet and passes the token to site B. Table 2 shows the final configuration of site C's OrderingQ.

3.0 Approaches to Verification and Validation of RMP

A formal proof of correctness for the original protocol specification exists [5], but we also wanted to ensure a high degree of fidelity between the specification and implementation of the protocol. To achieve this fidelity, we adopted a mode-based, tabular approach based on a variant of SCR-based tables [6] to express the protocol specification instead of the axiomatic approach in the original proof. Table 3 shows a small portion of the protocol specification tables for RMP. The first column shows the current mode. A mode is a superstate that encapsulates a larger set of specific states of an implementation [7]. While an implementation may change specific variables and thus move from state to state, the mode may remain unchanged until a major event and condition occur. Modes allow the specification to view states of the protocol machine at an appropriate level of abstraction for our analysis. Mode names in Table 3 include TokenSite (the site holds the token), NotTokenSite (the site does not hold the token), and Getting (the site holds the token, but must retrieve missing packets). The second column specifies the event which includes the arrival of a packet (data or acknowledgment (ACK)) or a time-out alarm. The third column specifies the condition under which a mode transition will occur given the event. In Table 3, we show conditions including checks for consistency of the Ordering Queue and checks to see if an incoming acknowledgment packet names this site as the new token site. We considered using condition tables [8] but our approach is currently sufficient for our protocol specification. The fourth column specifies the new mode if the event and condition are true. Finally, the fifth column specifies the action that takes place upon the mode transition. An action includes variable settings, conditions, and output events.

We used model checking to explore potential problems in the requirements mode model and used testing to explore suspect cases in the implementation. These tests helped verify that the implementation had the same behavior as the specification in specific cases. We tried several different specification methods for RMP including PVS [9], Murphi [10], SMV [11], and SPIN

Current Mode	Event/Alarm	Condition	New Mode	Action
NotTokenSite	Ack	OrderingQ consistent and named token site	TokenSite	-
NotTokenSite	Ack	Not named token site	NotTokenSite	-
NotTokenSite	Ack	OrderingQ inconsistent and named token site	Getting	Send Nack for missing packets
Getting	Data	OrderingQ consistent	TokenSite	-
TokenSite	Pass Alarm	OrderingQ consistent	Passing	Send Null Ack

Table 3: Fragment of RMP specification mode tables

[12]. We settled on a modified version of Murphi since (1) it was amenable to our tabular specifications and (2) includes temporal logic operators for verification of liveness, deadlock, and invariant properties of the specification. Tests were generated by hand from suspect cases and added to the test suite based on analysis of the Murphi models of the RMP specifications.

This type of approach to analysis played a major role in our effort even though we hoped that formal methods would reduce the need for testing. We discovered, however, that testing did not help us verify the protocol after its completion but rather it helped us to discover problems during the concurrent specification and implementation. We built a test scaffold for RMP by creating a low-level network stub and used testing as the vehicle for keeping our evolving implementation and specification in fidelity with each other. The code was annotated with debugging statements that produced a trace of events and conditions. Such traces were compared against the specification tables to validate the behavior of the implementation relative to the formal model. This approach proved to be very useful since the formal model helped us organize our test suite and provided an abstract model we could analyze.

We built the protocol specification and its implementation concurrently because pragmatic constraints of implementing the protocol had a feedback effect on the protocol specification. Performance requirements, programming language peculiarities, and other pragmatic aspects of the implementation forced us to consider changes to the requirements during implementation. We adopted an iterative approach to development because we expected these types of problems to occur. The design team built the first version of RMP with limited functionality to handle only nominal requirements of data delivery. This initial version did not handle off-nominal cases such as network partitions or site failures. Meanwhile, the V&V team concurrently developed the Murphi model of the requirements using the existing mode tables. Based on these requirements tables, the V&V team developed test cases to exercise the implementation. In a series of iterative steps, the design team added new functionality to the implementation while the V&V team kept the Murphi state model in fidelity with the implementation. This was done by generating test cases based on suspected errant or off-nominal behaviors predicted by the current model. If the execution of a test in the model and implementation agreed, then the test either found a potential problem or verified a required behavior. However, if the execution of a test was different in the model and implementation, then the differences helped identify inconsistencies between the model and implementation. In either case, the dialogue between both teams drove the co-evolution of the model and implementation.

4.0 Lessons Learned

Most of the problems found in the RMP specifications and implementation were caused by incomplete requirements where it was assumed that certain conditions could not occur but actually did occur in practice. Sometimes, the implementation was coded before the specification was updated if a pragmatic consideration made such a change expedient in the code. Other times, we explored solutions in the tables before coding it. Again, the testing between the specification and implementation during incremental development helped reveal these problems much earlier than if the process had been more linear.

4.1 The Perpetual Getting Problem

As shown in Table 3, a site will transition from NotTokenSite mode to TokenSite mode if the OrderingQ is consistent. If the OrderingQ is not consistent, then the site will enter the Getting mode while retrieving missing packets. Once the OrderingQ is consistent, the site will transition from Getting mode to the TokenSite mode. This fact was correctly specified in our mode tables, but the implementation was incorrect because a portion of code for the Getting mode did not check for consistency of the OrderingQ. The implementation livelocked in the Getting mode in the case of missing packets.

We were able to discover the problem during analysis for livelock modes using temporal assertions. A pessimistic analysis yielded potential off-nominal paths in the specification. Under ideal operating conditions of the protocol, no site should have to enter the Getting mode since no loss occurs under ideal conditions. Indeed, the problem was not discovered in testing on a Local Area Network where there was no loss of packets unless the network was congested (a rare condition). Subsequently, no sites ever entered the Getting mode to retrieve missing packets. The mode specifications do not explicitly model the loss of a packet, rather the condition of an inconsistent OrderingQ is an off-nominal behavior when a site becomes the token site. We constructed a test case for this scenario and found the problem in the implementation.

4.2 The Time-To-Live Problem

RMP relies on an unreliable IP Multicasting layer [13] in which packets have a time-to-live (TTL) field that controls their propagation in Wide Area Networks. At each router, the TTL field of a packet is decremented by 1 and checked to see if it is above or below the router threshold. If the TTL is above the threshold, the router forwards the multicast packet. If not, the packet is not forwarded. This allows control of the propagation of multicast packets to local, national, and world-wide distribution.

RMP extends the original TRP work by allowing for the initial formation and subsequent modifications to the token ring membership list during execution. RMP allows sites to join and leave the token ring dynamically. Our implementation, however, overlooked the fact that token rings sites can be local to one another (i.e., at low TTL values), but new sites can be very far away (i.e., at high TTL values). When the far site tries to join a ring, the far site will not see any messages due to their low TTL values. Subsequently, when the ring fails to pass the token to the far site. This failure will trigger a reformation of the ring to exclude the far site. This situation can repeat itself ad infinitum as long as the far site keeps trying to join the ring.

Time-to-live information was not included in the mode specifications. Therefore, no analysis of the formal model could have revealed this problem and we could not construct a test for this condition from the model. We feel, however, that this problem could have been detected during implementation when the design team needed to fill in the TTL field of the packets. The designers should have noted that the requirements are silent on how to fill-in the TTL field of any packet constructed. This silence invites a designer to make inconsistent assumptions about the behavior of the protocol machine.

4.3 The Leaving Ring Timestamp Problem

When a token site tries to leave a ring in a controlled fashion (i.e., rather than an abrupt site failure), it must wait until the token completes a cycle of the remaining ring sites before actually leaving the ring. The reason for this restriction is due to the fact that the departing site may hold packets that are missing at other sites. If the departing site leaves too soon, then some empty slots in the Ordering Queues of other sites cannot be filled.

The specifications incorrectly stated that a site may leave the ring when it has seen N timestamped packets where N is the number of site remaining on the token ring. The problem with this approach is that any intervening data packet can fill a timestamp slot causing the departing site to exit the ring before all remaining sites have acknowledged. We incorrectly assumed a one-to-one relationship between timestamps and acknowledgment packets. As a result, the ring is wedged in a livelock state because sites cannot fill some empty slots in their Ordering Queues.

The problem was found through direct analysis of the formal model and testing revealed the problem in the implementation. It took unusual conditions, however, to reproduce this problem in practice because the network had to be congested before the behavior appeared. The formal model produced a suspect path and the corresponding test produced a livelock condition. We feel that this problem was easily revealed by analysis of the formal model. In addition, the formal model helped structure exploration of test conditions during the resolution of the problem after its initial discovery.

5.0 Conclusions

We do not claim that RMP has been "verified and validated" to the extent that it is totally correct, rather that we have developed a technique that strengthens analysis and testing in the long-term development of our software. Short term problems did occur, but they helped us to evolve a specification model in high-fidelity with an implementation. Co-evolution of the formal specification model and the implementation was the most useful result of our study. Our technique allowed our two teams to structure their tests and other analysis activities. Their activities supported each other in the development of the implementation and refinement of the specifications.

In the future, we will continue to use RMP as a testbed problem and explore new specification and analysis techniques that complement incremental software development activities. We are continuing to evolve the specifications even though the software has been released in a Beta test version. This type of release scheme limits the use of RMP to non-critical projects and helps us explore operational problems. When a problem in operation does occur, we are using the mode tables to trace where the problem occurred. This has been useful in understanding problems, finding why problems were or were not detected earlier, and refining the specification incrementally.

References

[1] Luo, G., G. v. Bochmann, and A. Petrenko, Test Selection Based on Communicating Nondeterministic Finite-State Machines Using a Generalized Wp-Method, *IEEE Transactions on Software Engineering,* Volume 20, Number 2, February 1994, pp. 149-162.

[2] Sidhu, D. P. and T.K. Leung, Formal Methods for Protocol Tetsing: A Detailed Study, *IEEE Transactions on Software Engineering,* Volume 15, Number 4, April 1989, pp. 413-426.

[3] Montgomery, T., Design, Implementation, and Verification of the Reliable Multicasting Protocol, M.S. Thesis, West Virginia University, December 1994.

[4] Chang, J.M. and N.F. Maxemchuk, Reliable broadcast protocols, *ACM Transactions on Computer Systems,* Volume 2, Number 3, August 1984, pp. 251-273.

[5] Yodaiken, V. and K. Ramamritham, Verification of a Reliable Net Protocol, *Formal Techniques in Real-Time and Fault-Tolerant Systems,* January 1992, pp. 193-215.

[6] Heninger, K.L., Specifying software for complex systems: New techniques and their application, *IEEE Transactions on Software Engineering,* Volume 6, Number 1, January 1980.

[7] Jahanian, F. and A. K. Mok, Modechart: A Specification Language for Real-Time Systems, *IEEE Transactions on Software Engineering,* Volume 20, Number 12, December 1994, pp. 933-947.

[8] Leveson, N. G., M. P. E. Heimdahl, H. Hildreth, and J. D. Reese, Requirements Specification for Process-Control Systems, *IEEE Transactions on Software Engineering,* Volume 20, Number 9, September 1994, pp. 684-707.

[9] S. Owre, N. Shankar, and J. M. Rushby, User Guide for the PVS Specification and Verification System (Beta Release), Computer Science Laboratory, SRI International, 1991.

[10] D. Dill, A. Drexler, A. Hu, and C. Yang, Protocol Verification as a Hardware Design Aid, IEEE Conference on Computer Design: VLSI in Computers and Processors, IEEE Computer Society Press, October 1992.

[11] J. Burch, E. Clarke, D. Long, K. McMillan, and D. Dill, Symbolic Model Checking for Sequential Circuit Verification, IEEE Transactions on Computer-Aided Design, Volume 13, Number 4, April 1994.

[12] G. J. Holzmann and D. Peled, An improvement in formal verification, Proceedings of the 7th International Conference on Formal Description Techniques, FORTE 94, Berne, Switzerland, October 1994.

[13] Deering S., E., Multicasting Routing in Internetworks and Extended LANs, ACM SIGCOMM '88 Symposium, August 1988.