NORM Multicast Protocol Verification

Caleb Bowers

***Abstract*—**

I. Introduction

The past thirty years has seen significant improvement in how formal methods researchers approach verifying computer networking protocols. Traditionally, during development and implementation, the protocol would undergo extensive testing as the primary method of verifying that the performance of the protocol matched the design specification and the expected functionality. If the protocol behavior mirrored that of the design specification, then researchers and designers felt confident enough to label the protocol "verified," or at least stable enough for reliable use. This approach does not concretely verify the protocol, but does prevent obvious errors from existing and can come close to ensuring functionality in expected operational conditions. Extensive testing, however, only verifies the behavior of the specification for the specific domain tested, and does not logically verify the specification itself.

Extensive testing methods work fairly well for day to day unicast Internet protocols where the operating conditions are more predictable and the implementation less complex. There exists a gap, however, for multicast and broadcast networking protocols (hereafter, multicast[1]) and researchers have sought to develop more rigorous formal methods for these networking protocols.

In multicast environments the participants are often widely distributed and senders only know the group address, rather than the address of each participant (as is the case in unicast). The multicast communications paradigm requires more complex reliability and transmission mechanisms in order to ensure that the protocol maintains efficiency. Implementing these mechanisms leads to an explosion of state space, which further limits the efficacy of extensive testing, since that is often domain specific and is limited in time by the number of actual testable domains. Additionally, as previously mentioned, a specification may produce proper behavior amidst most operational conditions, but still not be logically sound, so a possibility for error exists somewhere in the behavioral domain for that protocol. Formally verifying these complex multicast mechanisms, therefore, remains necessary and is increasingly important as more communication becomes distributed and asynchronous across multicast protocols.

The focus of this paper is to replicate and build off of previous work in [1], which sought to verify an early draft version of the NACK Oriented Reliable Multicast Protocol (NORM) [2]. NORM is a fairly complex multicast protocol for a variety of reasons, namely its use of Forward Error Correction codes, its internal timing methods, and its NACK (negative acknowledgement) based reliability guarantees. Due to this complexity, the author of [1] focused only on two of the more approachable, yet still certainly complex, components of the protocol: its local and global round trip time calculation and its data repair transmission algorithms. For the purpose of this project, I only focus on replicating and updating the verification of the data repair transmission component of the current NORM protocol specification.

II. Related Work

In computer networking communication and protocol verification, we break communication into three main paradigms

- *unicast*: A sender transmits to a single receiver. Internet protocols operate on this model.
- *broadcast*: A sender transmits to all nodes on the network. This is how computers find printers on a local subnet.
- *multicast*: A sender transmits to a subgroup of nodes on the network.

Multicast communication enables a sender to transmit a message to a specific group address, or a multicast address [1]. Once the message is transmitted to this address, delivery of the message to every member of the group depends on the multicast network protocol. When specifying requirements for a multicast protocol, reliability is often provided by balancing the tradeoffs of feedback/retransmission strategies with the ability to scale to a large number of nodes across a wide network. If the protocol creates too much traffic, it will not scale well on its own as a communications protocol, nor will it be useful in larger networking contexts where

[1] It should be noted that multicast is a subset of broadcast. Broadcast entails a sender transmitting to all network participants, whereas multicast describes a sender transmitting to a *subgroup* of participants.

the multicast protocol must share resources with other communication protocols. While multicast protocols in design may be more complex than unicast communication paradigms, the approaches to their verification do not differ that significantly and a body of literature exists for general protocol verification dating back to the late 1970's [3]. What follows is a brief overview of some of the significant literature regarding the verification of both multicast protocols and communication networking protocols in general.

A. Early Protocol Verification

In [3], an early investigation is performed into the use of formal methods within the design phase of communication protocol development. The 1980's saw a proliferation of complex protocols across increasingly large and distributed networks. These protocols were often designed using extensive testing and narrative description (i.e., implementation use-cases), but design was rarely approached from a logical framework.

The authors' main contribution to verifying communication protocols was defining the necessary parts of a protocol that constitute a valid formal specification of the protocol in interest: service specification and protocol specification. A service specification is generally based on a set of service primitives and some examples include: *Connect, Disconnect, Send,* and *Receive*. A protocol specification "must describe the operation of each entity within a layer in response to commands from its users, messages from the other entities ... and internally initiated actions (e.g., timeouts)" (here an entity is a process or module local to each protocol implementation) [3]. The description of the operations of each entity within the protocol layer define the actual protocol when interacting across entities; therefore, early formal methods being developed for protocol verification focused primarily on the protocol *design* specifications (rather than *implementation* specific verification (i.e., code)), since this is concerned with the actual communication.

Additionally, [4], [5], and [6] provide further examples of early examinations of applying formal methods to network communication protocols.

B. Multicast Protocol Verification

As previously mentioned, multicast protocols can be significantly more complex than unicast protocols, since the communication originating from a sender can be destined for an arbitrary number of nodes at send time (as opposed to the lifetime of the packet in a unicast protocol). This complexity not only affects how to verify the underlying design of the communications protocol, but how to ensure certain desirable aspects of a communications protocol: security, wireless resilience, etc.

1) Early Attempts to Verify: Early examples of network protocol verification previously mentioned focused primarily on link level properties between fixed entities (e.g, sender and receiver) leveraging a fixed number of lower layer services and components. This approach does not enable the verification of protocols as they are used in the real world where communications happen between an exponential number of entities across complex protocols that make use of an arbitrary number of lower layer services, unicast or multicast. To account for the arbitrary nature of real protocol operational requirements, the researchers shifted from a strict model checking approach, to a technique that uses induction to prove correctness of protocol specifications and properties [7], [8] ,and [6].

2) Current State of Multicast Verification: Multicast protocols have seen increased use over the past decade as the proliferation of connected devices, improved wireless communication technologies, and increased connectivity has led to a networking environment more focused on communication between and within groups, rather than point to point communications characteristic of the early Internet. This increase in connectivity and devices has led to an emphasis on creating robust multicast protocols that are both reliable and secure. Wireless technologies further increase the need for security, especially within the multicast framework where there occurs less handshaking (in general) between communicating parties. In order to ensure these characteristics are present within a multicast protocol, there is a push to formally verify these protocols, specifically for group-oriented environments and wireless communications.

Prior to focusing on multicast protocol use in wireless settings, researchers turned to verifying additional security primitives for multicast network communication. The focus turned to verifying a system with an arbitrary number of components (as exemplified by streaming digital signature protocols) as a composition of security verified subprocesses carefully constructed to preserve the security properties of the entire system. This technique sufficiently proves that if each single process in a system satisfies a given single property, the composition of two or more *processes* satisfies the compositions of two or more *properties*. Examples using this composition method to demonstrate security satisfiability in multicast protocols are [9], [10], [11], and [12].

The emphasis within the field of multicast protocol verification quickly shifted to wireless settings as technological improvements enabled mass communication across the medium. Of particular interest is the function-

ality of wireless multicast protocols in sparse and poorly resourced networks, generally called edge networks. The authors in [13] provide an example of verifying a multicast protocol designed to operate within this networking context, MobiCast [14]. To verify this protocols, the authors develop a model of its functionality within the Prolog language and test this model for a variety of specification properties (i.e., the protocol operates in the way the design specifies) and safety properties (i.e., the protocol operates when it needs to).

Additional papers that cover wireless multicast vary in scope and focus, but a general lack of literature exists. Most wireless multicast research is focused on the development of new and efficient multicast protocols, leaving the verification until later. One additional example is [15], which examines the verification of a multicast protocol used to coordinate distributed mobile computing processes. The authors use the Calculus of Communicating Systems [5] and the Concurrency Workbench tool [16] to specify and verify their mobile computing multicast protocol.

III. Background

A. Real-Time Maude

For communication protocols time is inherently useful in constructing a communication paradigm and ensuring appropriate steps are taking in exchanging information. Since communicating nodes have no means of knowing whether a message will eventually arrive at its intended recipient, time plays a crucial role in helping nodes determine how to react to the receipt, or lack thereof, of a message/information when participating in a communication session. For example, if a node expects a reply to a previously sent message, it may set a timer to determine how long to wait for that reply before re-sending the message or moving ahead in a communication sequence [1].

Within the realm of computing there exists a subset of computing systems whose functionality depends on the passing of time in a real-world setting. These systems are referred to as *real-time* systems as their execution is tightly coupled to the passage of "real" time. Within this context, protocols constitute such a system, and NORM in particular heavily leverages and depends on the passage of time for proper functionality. In order to approach the verification of such a protocol, a modeling tool requires additional mechanisms to address the timing constraints a real-time system experiences. The author in [1] made us of the real-time specification language and analysis tool *Real-Time Maude* [17], [18]. Real-Time Maude extends the foundational specification language *Maude* [19], [20] based on a logic for modeling concurrent exchanges and evolution in distributed systems called re-write logic [21]. Real-Time Maude enables the user to specify exactly how the change in a concurrent system depends on time. What follows is a high -level view of the theory behind Real-Time Maude, since for the purposes of this project and paper, we simply need to know that Real-Time Maude can be used to specify, analyze, and verify real-time systems, such as NORM.

Real-Time Maude allows the user to analyze a specification of their system in the Maude language by:

- *Simulation*: User can specify a single behavior to simulate through the system specification
- *Exhaustive Search*: Real-Time Maude can search through all possible system states for infinite or bounded time from some initial state looking for safe or unsafe states (based on desired property)
- *Model Checking*: Using *Linear Temporal Logic Formulas* Real-Time Maude verifies that from some given initial state the system satisfies the formula for all states or up to some time bounded set of states.

1) Rewriting Logic: Rewriting logic [21] is particularly well-suited for specifying and analyzing concurrent and distributed systems. A specification in rewrite logic (or a *rewrite theory*) is a tuple $\mathcal{R} = (\Omega, E, L, R$. Where Ω is an equational signature and E represents a set of equations and membership axioms [1]. L is a set of labels and R is a set of conditional and unconditional rewrite rules in form:

- Unconditional: $l : t \rightarrow t'$
- Conditional: $l : t \rightarrow t' \; \boldsymbol{if} \; cond$

Where $l \in L$, t and t' are terms in a set of well formed terms in the signature Ω and *cond* is a conjunction of rewrite conditions.

Rewrite logic employs deduction steps in order to determine if given a rewrite theory, this theory satisfies some condition in time t for the following deductive rules: Reflexivity, Equality, Transitivity, Congruence, and Replacement. Please consult [1] Section 2.1.1 for a formal definition of these deductive rules.

2) Using and Analyzing Real-Time Maude: Maude leverages rewrite logic to create specifications for systems in order to perform formal analysis and verification the state behaviors of those system. Likewise, Real-Time Maude further extends the theory or rewrite logic to encompass time dependent systems and executions by including syntax to specify the time constraints on a Maude command. A simulation of a specific behavior of a system can now be bounded for set time or can be

simulated with out a time constraint, allowing Maude to find unsafe states/behaviors. Similarly, Real-Time Maude enables time specifications for state space search methods and for evaluating linear temporal logic formulas. This enables users to evaluate time itself as a possible failure cause for system behaviors, which, as is the case for NORM, directly impacts functionality of a real-time system.

B. NORM

1) NORM Overview:

2) NORM Specification and Model:

IV. NORM Data and Repair Transmission Component

A. Specification

1) Replication:

2) Update Specification:

B. Analysis

References

[1] E. Lien, "Formal modelling and analysis of the norm multicast protocol using real-time maude," 2004.

[2] J. P. Macker, C. Bormann, M. J. Handley, and B. Adamson, "NACK-Oriented Reliable Multicast (NORM) Transport Protocol," RFC 5740, Nov. 2009. [Online]. Available: https://rfc-editor.org/rfc/rfc5740.txt

[3] G. V. Bochman and C. A. Sunshine, "Formal methods in communication protocol design," *IEEE Transactions on Communications*, 1980.

[4] G. J. Holzmann, *Design and Validation of Computer Protocols*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1991.

[5] R. Milner, *Communication and Concurrency*. Prentice-Hall, 1989.

[6] M. Baptista, S. Graf, J.-L. Richier, L. Rodrigues, C. Rodriguez, P. Veríssimo, and J. Voiron, "Formal specification and verification of a network independent atomic multicast protocol." 01 1990, pp. 345–352.

[7] S. J. Creese and J. Reed, "Verifying end-to-end protocols using induction with csp/fdr," in *Parallel and Distributed Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 1243–1257.

[8] J. R. Callahan and T. L. Montgomery, "Verification and validation of a reliable multicast protocol," NASA, Tech. Rep., 1995.

[9] R. Gorrieri, F. Martinelli, and M. Petrocchi, "Formal models and analysis of secure multicast in wired and wireless networks," *Journal of Automated Reasoning*, vol. 41, no. 3, pp. 325–364, Nov 2008.

[10] J. E. Martina and L. C. Paulson, "Verifying multicast-based security protocols using the inductive method," *International Journal of Information Security*, vol. 14, no. 2, pp. 187–204, Apr 2015. [Online]. Available: https://doi.org/10.1007/s10207-014-0251-z

[11] G. Bella, L. C. Paulson, and F. Massacci, "The verification of an industrial payment protocol: The set purchase phase," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02. New York, NY, USA: ACM, 2002, pp. 12–20. [Online]. Available: http://doi.acm.org/10.1145/586110.586113

[12] M. Archer, "Proving correctness of the basic tesla multicast stream authentication protocol with tame." NRL, 2002.

[13] M. Borujerdi and S. Mirzababaei, "Formal verification of a multicast protocol in mobile networks." 01 2004, pp. 270–273.

[14] C. L. Tan and S. Pink, "Mobicast: A multicast scheme for wireless networks," *Mobile Networks and Applications*, vol. 5, no. 4, pp. 259–271, Dec 2000. [Online]. Available: https://doi.org/10.1023/A:1019125015943

[15] G. Anastasi, A. Bartoli, N. De Francesco, and A. Santone, "Efficient verification of a multicast protocol for mobile computing," *The Computer Journal*, vol. 44, 12 2000.

[16] R. Cleaveland and S. Sims, "The ncsu concurrency workbench," in *Computer Aided Verification*, R. Alur and T. A. Henzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 394–397.

[17] The real-time maude webpage. [Online]. Available: http://heim.ifi.uio.no/peterol/RealTimeMaude/

[18] P. C. Ölveczky and J. Meseguer, "Specification and analysis of real-time systems using real-time maude," in *Fundamental Approaches to Software Engineering*, M. Wermelinger and T. Margaria-Steffen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 354–358.

[19] The maude webpage. [Online]. Available: http://maude.cs.illinois.edu/w/index.php/The_Maude_System

[20] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada, "Maude: specification and programming in rewriting logic."

Theor. Comput. Sci., vol. 285, pp. 187–243, 01 2002.

[21] J. Meseguer, "Conditional rewriting logic as a unified model of concurrency," *Theor. Comput. Sci.*, vol. 96, no. 1, pp. 73–155, Apr. 1992. [Online]. Available: http://dx.doi.org/10.1016/0304-3975(92)90182-F