Network Multicast Protocol Verification: A Literature Review

Caleb Bowers

10-31-2019

Abstract

The us of multicast protocols continues to increase across a variety of networks: mobile, wireless, and local networks. These protocols are generally more complex than standard unicast protocols, especially when they are designed to be efficient and reliable. This complexity prevents designers from understanding the limitations or failure modes of the protocols and can have disastrous consequences in implementation. In order to understand the current state of verification for multicast protocols, I summarize several historical multicast verification papers in addition to more modern examinations of verification for these protocols that extend to security and reliability. I then summarize an example of attempts to formally specify and verify a complex multicast protocol that provides efficiency and reliability. These works represent only a sample of the available literature, but do capture the efforts undertaken to verify multicast protocols, which is becoming increasingly popular, but has been outstripped by the surge in complexity of multicast protocols.

CONTENTS

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'.' 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.

[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 goal of this paper is to review several of the primary literature sources and seminal results that have contributed to the field of formal multicast network protocol specification. What follows is a sampling of the multicast verification work and is no way exhaustive. The literature sampled ranges from fundamental theory and the development of formal methods for multicast protocol verification, to highlighting the development and workings of the tools used in practice to verify multicast protocols.

II. BACKGROUND

A. Network Protocols

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 the multicast protocol must share resources with other communication protocols. The focus of this literature review is to provide an understanding of the literature that has contributed most notably to the development of formal methods for networking protocols and specifically, multicast protocols.

III. Literature Overview

A. Early Protocol Verification

In [2], 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. As the use of these protocols increased, the variety of different implementations for a given protocol also increased, which in turn, increased the complexity of testing the protocol in order to ensure specification compliance. This environment created the need for more formalized design and implementation verification methods.

The authors begin their work by defining what constitutes a protocol specification. A protocol specification, broadly speaking, consists of two parts: 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) [2]. 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, since this is concerned with the actual communication (implementation verification can be viewed as the more common program verification problem). Protocol design verification methods leveraged general formalisms such as Petri nets, state analysis, and verification programming languages [3].

Finally, when actually attempting to verify a protocol, early verification in practice could be classified as either reachability analysis or program proofs. Reachability analysis exhaustively analyzes all of the possible interactions two or more entities may experience when using a given communication protocol. To verify, a global state is defined that composes the states of the protocol and connected entities. From a start state, all possible transition states are generated and the possible combinations of these form new global states, which need to be tested for conformance to the protocol design specifications. Program proofs for protocols pose some difficulty since protocols can be unreliable (nondeterministic) and are often concurrent when multiple entities are involved, which prevents entities from sharing the same set of variables and causing the proof variable space to explode. Often proofs of protocols occur upon topologies: a property is shown true for some subset and then inducted for *n, n+1* entities, an example of this will be examined later.

To avoid an explosion in the state space, several methods are available specific to protocol design verification. Verifiers may focus on only a portion of the specification that is often most critical to correct protocol function. Additionally, verifiers may reduce those state transitions that are indivisible into a single transition (e.g., wrapping a packet and then sending it could be viewed as one state). A protocol may also be decomposed into its sublayers/phases of operation to simplify the description and verification of properties relevant to those sublayers. By separately proving correctness for each layer, the protocol as a whole is then verified. Assertions (i.e., predicates) can be defined for a set of states, so verifying these assertions enables the verification for all possible states (as long as assertions exist to cover all possible states). State space can further be reduced by combining assertions with a focused search approach. This reduces the state space to those states that can be potential candidates for some defined assertion (e.g., deadlock).

The paper concludes by providing examples of early adopters of protocol verification design strategies, namely, the ARPANET Initial Connection protocol and the Cyclades transport protocol. The authors then set the expectation that more fundamental formal methods will be developed that can be extended to more complex and actually implemented protocols. In addition to this paper, another early paper introducing formal verification as a viable, automatic method for protocol design is [4]. The papers that follow and others attempting to formally verify protocols benefit extraordinarily from this early

work. Without this contribution, additional development of formal methods for communication protocols would have been significantly slowed, since the authors demonstrated that protocol specifications can be translated into the language of verification and implemented in a variety of proof solvers. The actual methods researchers use is detailed in the following subsections.

B. Early Multicast Protocol Verification

Motivation for [5] came from the state of protocol verification at the time of publication, which 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 protocol operational requirements, the authors shift from a strict model checking approach, to a technique that uses induction to prove correctness of protocol specifications and properties. They implement this method using FDR, a software package developed by Formal Systems Ltd. that is based on the theory of Communicating Sequential Processes (CSP). To provide inductive capability for an arbitrary number of components, the authors extended the original CSP theory by using lazy abstraction[2] and model checking components that do not exhibit inductive behavior in order to verify all properties of the given protocol[3].

Layered protocols lend themselves to examination as finite state machines (FSM), since each layer can be expressed as an FSM and the correctness of each layer's FSM impacts the following layer's FSM correctness. While FSMs account for the layered nature of protocols, they fail to address the unbounded network topology that can be encountered by an operational protocol. To account for unbounded topologies, the authors modeled topologies as "action systems" operating processes that interacts with their environment through discrete events. This model enables a system to reduce a process to a sequence of events and this sequence itself can be verified. The authors used this as motivation to select CSP/FDR as their formal verification method. CSP/FDR are formalisms that combine programming languages (CSP) and finite state machines (FDR) in order to verify a system of systems [6].

The authors test their induction method on the RSVP multicast reservation protocol that operates on IP networks [8]. RSVP multicasts from information sources to receivers by "transmitting along a number of intermediate links shared by "downstream" nodes." RSVP then creates and maintains the reservations along each link of the previously defined multicast route. The route is finite from a "downstream" node to the original source node and can contain any number of links or nodes in between.

By employing Lazy Abstraction the authors reduce an unbounded state space to a finite space by abstracting away the unbounded components of the multicast network. They achieve this by viewing the protocol from the perspective of a downstream node as protocol function/decisions move along the upstream link towards the sender.

The authors' model examines the traffic reducing nature of RSVP (intermediate nodes can automatically respond to requests that they have already seen) by employing FDR to perform a *structural induction* that establishes the properties for the end-to-end nature of RSVP by a series of refinement checks. These refinement checks actually construct the inductive proof. Having established the validity of the inductive proof for any arbitrary multicast route between nodes, the authors proceed to evaluate this method on a multicast network in the form of a tree: the source as the root node and the receivers as the leaf nodes. They construct a general model of a network node, apply the inductive properties, and establish the communication primitives for root, leaf, and intermediate nodes. They are able to cleverly reduce an intermediate node to a source node for all downstream communications from that intermediate node. In doing so, they simplify the necessary verification steps. It is important to note that the downstream communication properties for arbitrary routes and nodes are verified through induction, which is the main contribution of this paper. This means that for each communication property, the authors establish a base case and then a correlative inductive case. RSVP also seeks to minimize network

[2]Lazy abstraction compresses the abstract-check-refine process into a single process to "continuously builds and refines a single abstract model on demand, driven by the model checker, so that different parts of the model may exhibit different degrees of precision, namely just enough to verify the desired property." [6]

[3]For a fuller summary of inductive methods applied to multicast please refer to [7], which provides an overview of the history of inductive methods applied to multicast protocol verification and a comprehensive framework for implementing inductive methods.

traffic along a node's upstream path, and the authors verify this property by simply abstracting each node to be a receiver that only sends reservation requests when the node sees a new request. Applying this abstraction to all downstream channels, the property can be verified iteratively for each downstream node.

To conclude their paper, the authors highlight that the inductive approach may work locally for a node, but fail for the entire system. As an example, delay on a per node basis may be within the acceptable specifications, but in the aggregate could fail delay requirements for the system as a whole. So, while certain end-to-end properties can be more easily verified by induction, not all lend themselves to this approach. The authors assert that the main benefit of their work is to verify the existence (or absence) of end-to-end deadlock or divergence behavior of a complex protocol.

Additional examples of early multicast protocol verification include [9] and [10], which focus on smaller scale multicast protocols applied to specific Internet contexts. The authors in both papers sought to wed more closely the design process of a protocol and its formal verification in an effort to prevent egregious errors from occurring later during testing. While this stood as a noble goal, the practice of combining design and verification remains fairly uncommon. Despite the authors' designs for a future of tightly coupled design and verification not becoming a reality, their early work enabled formally specifying components of multicast protocols to be an approachable research area, even if after the initial protocol design period.

C. Recent 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 hand-shaking (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. It should be noted that the majority of work conducted in this area is more focused on the formal verification of specific multicast protocols using existing formal methods, rather than developing new formal methods and approaches for the multicast communications model in general.

The following sections will highlight several papers that examine the verification of different properties of multicast communication: security primitives, wireless communication, agreed timing, and reliability. Less summarizing detail will be afforded to each paper than in previous sections, since what follows is informational, rather than providing background or a narrative of the development of the field of multicast verification. That being said, the reader should have a sufficient understanding of the methods and results for each paper.

1) Security: Perhaps somewhat surprisingly, prior to focusing on the verification of multicast communication within wireless settings, the security guarantees of multicast protocols received early attention in [11] for both wired or wireless domains. The authors specifically focused on the security of digital streams across multicast streaming protocols. These are of particular interest, because multicast data streams see significant use in file sharing, live-streamed broadcasts, massive online video games, and even the pushing of software updates. So, while this paper is not particularly concerned with verifying the communication properties of multicast protocols, it is concerned with verifying the implementation of security protocols and primitives (authenticity, integrity, and confidentiality) within a multicast communications framework.

In order to verify a variety of security properties, the authors focus on 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. More plainly, the compositional principle provides sufficient proof to conclude that if each single process satisfies a single property, the composition of two or more *processes* satisfies the compositions of two or more *properties*. The authors employ this approach to verify a property for integrity (a stability against packet modification) and a secrecy property that requires the stream content to remain unknown to everybody but the sender and the intended set of receivers. They verify the presence of these properties for the Gennaro-Rohatgi protocol

(a stream signing protocol) [12], the Efficient Multi-chained Stream Signature protocol (EMSS) [13], the μTESLA protocol (micro Timed Efficient Stream Loss-tolerant Authentication to provide authenticated wireless broadcast for sensor networks [14]), and a secret key multicast group distribution protocol, often referred to as the N Root/Leaf pairwise protocol [15].

Since the authors' primary concern is protocol integrity and secrecy, they consider two integrity properties, one untimed for EMSS and one timed for μTESLA and a secrecy property for the N Root/Leaf pairwise protocol. To formally verify these properties the authors employ a schema called Generalized Non Deductibility on Compositions (GNDC). This schema compares expected system behavior with a system behavior modified by some malicious process trying to interfere with expected execution. If these behaviors appear the same, then the intruder has had no real affect on the operation of the system and protocol, thus verifying the property of interest. Using this schema and approach, the authors verify that both EMSS and μTESLA enjoy integrity for an arbitrary number of multicast receivers (μTESLA must also have timed secrecy on its secret keys in order to ensure timed integrity). Additionally, the authors conclude that the N Root/Leaf pairwise protocol preserves the secrecy of a given message m containing a given key k.

In performing this analysis, the authors demonstrate the verification of multicast security protocols that can be implemented in real world systems, and thus, provide guaranteed security (for the formal definition of security) throughout the life of their operation for an arbitrary number of participants and components. Their work inspired additional formal multicast security verification as the wireless networking age came into fruition: [7], [16], and [17].

2) Wireless: Wireless and mobile networks have seen an exponential increase in usage and viability over the past 15 years. This medium has enable additional connectivity and provided an environment ripe for multicast protocol use. Wireless networks often coordinate with a base station, which acts as a hub to the greater Internet. Working within this model, group-based control and coordination messages often need to be disseminated across a wireless network, a communication paradigm ideally suited for multicast.

In [18], the authors examine verification for multicast protocols in wireless mobile networks, specifically for edge wireless networks connected to base stations. Here the authors formally specify the MobiCast protocol[19] within the Prolog language[4], which enables semi-automated reasoning to be applied to the specification in order to verify some set of properties.

In order to verify the protocol, the authors examine some randomly generated topology relevant to the cell network model. Using this topology, they perform graph theoretic analysis to build a Steiner tree, which defines a path between the subset of cells (represented by the base stations). Upon this path, the authors then define the high-level Prolog definitions for the MobiCast protocol.

Combining graph theoretic algorithms and the Prolog specifications, the authors created a formal representation of the MobiCast protocol for some generated topology. The authors generate topologies that conform to some constraints that guarantee a valid topology for the MobiCast protocol, so they can focus solely on verifying MobiCast, rather than having to account for the influences of the network topology. The authors then seek to verify a variety of properties including the termination property (given a certain input, will this program terminate?) and response property (Given a certain input, if the program terminates, what is the output of the program?).

After completing the formal specification, the authors claim to find and resolve several inconsistencies by making minor variations to MobiCast through relating the inconsistent protocol sections with contradictory properties. If one section of the protocol was working to ensure some property, then the authors were able to tweak other parts to *not* work towards that property. The authors indicated that this could be used for future work in verifying the co-existence of two different protocols (e.g., TCP/TIP), but I was unable to find such a work by these authors. Formal verifications of TCP do exist [20], but are beyond the scope of this literature review. Additionally, despite the authors claiming that they were able to resolve the inconsistencies within the MobiCast protocol, they do not actually provide the reader with any information on what these inconsistencies may be. There is a decent bit of hand-waving performed on the actual results the authors claim to have produced, with them substituting long Prolog definitions for actual substantive

[4]MobiCast is a multicast protocol suitable for a network consisting of small cells with mobile nodes within those cells. The base stations of these cells have a wired back-plane connected to a high speed network that can send multicast messages from one cell to another. This is somewhat the reverse of what is mentioned above, rather than a base station multicasting to its group, a node within a group (cell) would like to multicast to multiple other cells/nodes within cells.

results. This paper does, however, provide a process and framework for producing specifications within Prolog.

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 [21], which examines the verification of a multicast protocol used to coordinate distributed mobile computing processes. The authors use the Calculus of Communicating Systems [22] and the Concurrency Workbench tool [23] to specify and verify their mobile computing multicast protocol.

D. NACK Oriented Reliable Multicast

This last section details the verification of the NACK Oriented Reliable Multicast protocol (NORM) [24], which is the primary focus of [1], the doctoral thesis of Elizabeth Lien. This is also the inspiration for my semester long project. Within her thesis she examines two crucial components of NORM: the data and repair transmission component (reliability) and the estimation of greatest round-trip time (ensure common time-out bases). She specified the components and verified them using the verification language Real-Time Maude (hereafter, simply called Maude).

1) Round Trip Time Estimation: By specifying the greatest round-trip time (GRTT) component of the receiver group for an instance of NORM, the author was able to take the procedures described in the informal specification and model them fairly straightforwardly in Maude. She abstracted away any informal specifications present in the actual code that did not pertain to the GRTT estimation, which greatly reduced her state space.

In her analysis she defined two initial states for two different network topologies. The first is simple: using only four nodes, the GRTT is easily found since the receiver round-trip times (RTTs) are the same for all nodes in the receiving group. The second initial state has six nodes with several different RTTs, so the GRTT is more complicated to calculate. Her analysis concluded the following:

- the sender was able to calculate a GRTT estimate close to the recorded highest RTT value in the receiver group;
- the sender then multicasted the GRTT estimate to the receivers;
- an inconsistency with updating the GRTT versus the RTT was found within the GRTT estimation algorithm.

These results show how an algorithm can work fairly well in practice, but verification shows corner cases where it may break.

2) Reliability: Modeling the data and repair transmission component in Maude was quite complex due to inconsistencies and ambiguities within the informal specification for NORM. This forced the author to make interpretive decisions regarding what the specification actually meant for the ambiguous sections. While this did bias her results towards her own interpretation, she did communicate with the original designers in an effort to clarify their informal specifications. She conducted her analysis for two initial states with the same topologies she used for the GRTT estimation. These initial states, apart from different topologies, differ only in router delay.

Her analysis for the data and repair transmissions is fairly inconclusive, which is primarily the result of incomplete informal specifications. The informal specification did not provide sufficient information for a variety of the repair components, so the behavior that resulted from her verification analysis both verified the repair property (the sender repaired lost packets), but also failed to verify. The failures resulted from a simple increase in traffic load causing the network to drop the repair requests. This indicates that these packets are not afforded the necessary priority within the NORM protocol that they actually require for proper functioning.

From her results, we can see that both formally specifying and verifying a complex communication protocol can be almost impossible, especially when the informal specification lacks sufficient detail. This itself creates a development process heavily reliant on large scale iterative testing to ensure operational guarantees, rather than designing with verification in mind. We can see that significant work remains to both formally specify and then verify the NORM protocol, but her thesis provides a good foundation from which to build.

IV. Conclusion

This sample of papers examines the state multicast protocol verification and some of the historical works that helped lay the groundwork for communication protocol verification. With the increase in need for the use of these protocols, work investigating their verification

will also grow. The flexibility of the NORM protocol for wireless and mobile networks enables its use in a variety of contexts, but research into its formal verification must be performed in order to ensure the protocol is logically sound regardless of it operational context or topology.

REFERENCES

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

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

[3] C. A. Petri, "Communication with automata," 1966.

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

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

[6] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre, "Lazy abstraction," in *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '02, (New York, NY, USA), pp. 58–70, ACM, 2002.

[7] J. E. Martina and L. C. Paulson, "Verifying multicast-based security protocols using the inductive method," *International Journal of Information Security*, vol. 14, pp. 187–204, Apr 2015.

[8] R. T. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification." RFC 2205, Sept. 1997.

[9] J. R. Callahan and T. L. Montgomery, "Verification and validation of a reliable multicast protocol," tech. rep., NASA, 1995.

[10] 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.," pp. 345–352, 01 1990.

[11] 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, pp. 325–364, Nov 2008.

[12] R. Gennaro and P. Rohatgi, "How to sign digital streams," vol. 165, pp. 110–116, Information and Computation, February 2001.

[13] A. Perrig, R. Canetti, J. D. Tygar, and Dawn Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000*, pp. 56–73, May 2000.

[14] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. E. Culler, "Spins: Security protocols for sensor networks," *Wireless Networks*, vol. 8, pp. 521–534, Sep 2002.

[15] E. J. Harder and D. M. Wallner, "Key Management for Multicast: Issues and Architectures." RFC 2627, June 1999.

[16] 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*, CCS '02, (New York, NY, USA), pp. 12–20, ACM, 2002.

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

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

[19] C. L. Tan and S. Pink, "Mobicast: A multicast scheme for wireless networks," *Mobile Networks and Applications*, vol. 5, pp. 259–271, Dec 2000.

[20] M. A. S. Smith, *Formal Verification of TCP and T/TCP*. PhD thesis, Cambridge, MA, USA, 1997. AAI0599243.

[21] 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.

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

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

[24] J. P. Macker, C. Bormann, M. J. Handley, and B. Adamson, "NACK-Oriented Reliable Multicast (NORM) Transport Protocol." RFC 5740, Nov. 2009.