Network Multicast Protocol Verification: A Literature Review

Caleb Bowers

10-31-2019

Abstract

Multicast protocols continue to increase in use 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 implementers 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 papers, modern examinations of multicast protocols for security and reliability in both wireless and wired contexts, and provide an example of an undertaking to formally specify and verify a complex multicast protocol that provides efficiency and reliability. These works represent a sample of the available literature, but do capture the efforts undertaken to verify multicast protocols, which is becoming increasingly popular, but has not seen as much popularity as the protocol complexity increases.

Contents

1 Introduction **1**

2 Background **2**
2.1 Network Protocols 2

3 Literature Overview **3**
3.1 Early Protocol Verification 3
3.2 Early Multicast Protocol Verification 4
3.3 Recent Multicast Verification 6
3.3.1 Security 6
3.3.2 Wireless 8
3.4 NACK Oriented Reliable Multicast 9
3.4.1 Round Trip Time Estimation 9
3.4.2 Reliability 9

4 Conclusion **10**

References **10**

1 Introduction

The past thirty years has seen significant improvement in how formal method and verification 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 that specific tested domain, 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

[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

mechanisms of reliability and transmission to ensure efficient performance of the protocol. 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 not be logically sound, so a possibility for error exists somewhere in the behavioral domain for that specification. Formally verifying these complex multicast mechanisms, therefore, remains necessary and is increasingly important as more communication becomes distributed and asynchronous.

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.

2 Background

2.1 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 communication protocols and specifically, multicast communication protocols.

3 Literature Overview

3.1 Early Protocol Verification

In [2] we see an early investigation into the use of formal methods in the designing of communication protocols. The 1980's saw a proliferation of complex protocols across an 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. This further escalated the complexity of the protocol to ensure design compliance, which created the need for more formalized design and implementation verification methods.

In [2], 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 (and 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 in 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, we define a global state composed of 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 on protocols occur on topologies: a property is shown to be true for some subset and then inducted for *n, n+1* entities.

To avoid an explosion in the state space, several methods are available specific to protocol design verification. Verifiers may focus on verifying only a portion of the specification, that which 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 verifying each layer, the protocol can then be verified. Assertions that are predicates can be defined for a set of states and 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 combing assertions with a focused search approach that reduces the state space from a global view to those that can be predetermined as potential states for some 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. Building off this expectation the next section will examine how the literature has progressed to address the inherently more complex operation of multicast protocols, which exhibit the more challenging behaviors to verify (e.g., concurrent transmissions/connections, unreliability, etc.).

A seminal paper more rigorously classifying formal verification as a valid method for protocol verification as an automatic method 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.

3.2 Early Multicast Protocol Verification

The motivation for [5] came from the state of protocol verification at that time. As briefly alluded to in the previous section, formal specification and verification of communication protocols 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.

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 a software package developed by Formal Systems Ltd. called FDR 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.

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 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. These topologies at this time were modeled as an action system operating a process which interacts with its environment through discrete events. This 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).

The authors test their induction method on the RSVP multicast reservation protocol that operates on IP networks [7]. 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 in the behavior specification of the protocol as it moves along the upstream link towards the sender.

The creation of the model in this paper examines the traffic reducing nature of RSVP (intermediate nodes can automatically respond to requests that they have already seen. They use FDR to perform a *structural induction* that establishes the properties for the end-to-end nature of RSVP by a series of refinement checks that actually comprise the inductive proof. Having established the validity of the inductive proof fo any arbitrary multicast route between nodes, the authors proceed to evaluate this method on a multicast network in the form of a tree with the source as the root node and the receivers forming the leaf nodes of the tree. They construct a general model of a network node, apply the inductive properties, and establish the communication primitives for root, leaf, and intermediate nodes and somewhat cleverly reduce an intermediate node to that of a source for downstream communications, which simplifies 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 correlative inductive case. Additional properties such as minimizing network traffic along a node's upstream path is able to be

[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]

verified simply as abstracting each node to be a receiver that only sends reservation requests when the node sees a new request. By applying this abstraction to all downstream channels, we can verify this property for each downstream node iteratively.

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, they provide that 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.

3.3 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 towards communication between groups and within groups, rather than point to point communication characteristic of the bourgeoning 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 less hand-shaking (in general) between communication parties occurs. 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, reliability, and congestion control. Less 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.

3.3.1 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 [8] for either wired or wireless media. The authors specifically focused on the security of digital streams across multicast streaming protocols. These are of particular interest, because a significant amount of file sharing, live-streamed broadcasts, massive online video games, and even the pushing of software updates occur via multicast data streams. So, while this paper is not particularly concerned with verifying the communication properties of multicast protocols, it is concerned with verifying the application of security protocols and primitives (authenticity, integrity, and confidentiality) to a multicast stream.

In order to verify a variety of security properties, the authors focus on verifying a system with an arbitrary number of components (as exhibited by streaming digital signature protocols) as a composition of security verified subprocesses constructed carefully 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) [9], the Efficient Multi-chained Stream Signature protocol (EMSS) [10], the μTESLA protocol (micro Timed Efficient Stream Loss-tolerant Authentication to provide authenticated wireless broadcast for sensor networks [11]), and a secret key multicast group distribution protocol, often referred to as the N Root/Leaf pairwise protocol [12].

As previously mentioned, the authors are concerned with the analysis of integrity and secrecy properties. They consider two integrity properties, one untimed for EMSS and one timed for μTESLA. For N Root/Leaf pairwise protocolthey consider consider the secrecy property. To formally verify these properties a schema called Generalized Non Deductibility on Compositions (GNDC) is used. 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, and the property of interest is verified. Using this framework the authors verify that both EMSS and μTESLA enjoy integrity for whatever 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 verification work and laid the ground for the ripening wireless networking age.

3.3.2 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, where this base station acts as a hub to the greater Internet. Working within this model there are often group based control and coordination messages needing to be disseminated to the wireless network, which is a communication paradigm ideally suited for multicast.

An example of verification for multicast protocols in wireless mobile networks specifically for edge wireless networks connected to base stations is examined in [13]. Here the authors formally specify the MobiCast [14] protocol[3] within the Prolog language, 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. From this topology, they perform some graph theoretic analysis to build a Steiner tree, which will define 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 have created a formal representation of the MobiCast protocol for some generated topology. In order for the verification of MobiCast to be isolated from the verification of the topology as a viable topology for MobiCast to operate within, the authors generate topologies that conform to some constraints that guarantee a valid 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 were able to find several inconsistencies and resolve them by making minor variations to MobiCast by relating the different 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, but formal verification of TCP do exist [15], but are beyond the scope of this literature review.

[3]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.

3.4 NACK Oriented Reliable Multicast

This last section details the verification of the multicast protocol NACK Oriented Reliable Multicast protocol (NORM) [16], which is the primary focus of the doctoral thesis [1]. This is also the inspiration for my semester long project. Within her thesis she examines two crucial components: 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).

3.4.1 Round Trip Time Estimation

In 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 reduces her state space.

In her analysis she defined two initial states for two different network topologies. The first is simple and contains only four nodes and the GRTT is easily found since the receiver round-trip times (RTTs) are the same. 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, and
- 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.

3.4.2 Reliability

Modeling the data and repair transmission component in Maude was quire complex due to inconsistencies and ambiguities within the informal specification for NORM. This led the author to have to make interpretive decisions regarding what the specification actually meant for the ambiguous sections. While this will bias her results towards her interpretation, communication with the original designers was undertaken in an effort to clarify. The actual specification for this part of the protocol is quite lengthy and is left to remain in her thesis, but 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 do so. The failures resulted from a simple increase in traffic load causing the network to drop the repair requests, which seems to indicate these packets are not afford the necessary priority they require.

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 is lacking in 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.

4 Conclusion

This literature review examined a sample of papers from the world of 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 it to be used extensively 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] R. T. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification." RFC 2205, Sept. 1997.

[8] 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.

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

[10] 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.

[11] 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.

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

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

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

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

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