



DOI:10.1145/3706572

The accelerated migration to advanced services will be accompanied by unprecedented complexity, and security and reliability concerns that must be addressed by the network-engineering and formal-methods communities.

BY DAVID BASIN, NATE FOSTER, KENNETH L. MCMILLAN,
KEDAR S. NAMJOSHI, CRISTINA NITA-ROTARU,
JONATHAN M. SMITH, PAMELA ZAVE, AND LENORE D. ZUCK

It Takes a Village:

Bridging the Gaps between Current and Formal Specifications for Protocols

NETWORK PROTOCOLS ARE fundamental to the functioning of modern society, as more and more services are provided online, including critical infrastructure. Historically, these services have mostly been implemented with wired networks that connect conventional computers. But with the emergence of wireless and cellular technologies, their reach has

extended to Internet of Things (IoT) devices and sensors, robotic systems, and autonomous vehicles. The next generation of cellular technologies, often referred to as *NextG*, will further expand the set of services that can be delivered over a network, including real-time remote healthcare, industrial automation, precision agriculture, smart infrastructure, holographic communication, space-based communication, and more.

We are thus facing a future where a vast number of complex and critical services will emerge, and it is unrealistic to expect them to function correctly and be resilient to attacks in every scenario. It is often assumed that the well-understood Internet protocols that have been used are mostly correct and secure. This assumption is, however, false: Vulnerabilities in old protocols, such as DNS, are frequently discovered and exploited. A review of the Common Vulnerabilities and Exposures (CVE)TM database reveals that many vulnerabilities stem from poorly specified and poorly understood specifications, leading to implementations that are prone to attacks.

Despite limited use of formal methods (FM) in the design of its protocols, the Internet mostly works, for some definition of “works.” With the transition to NextG technologies, significantly higher levels of assurance will be needed, especially for net-

» key insights

- Experience shows that formal specification can provide significant practical benefits for network protocol development by enhancing clarity, trustworthiness, and productivity.
- Adoption of formal methods has been hindered not by lack of scalability but by cultural gaps that result in tools and methods that do not align with engineering practice.
- We need collaboration between formal methods researchers and the networking community to capture the engineering culture of particular disciplines in usable formal languages, tools, and methodologies.

IMAGE BY JOZEF MICIC



works supporting safety-critical systems. The sheer number of emerging technologies and the potential risk to critical infrastructure and human lives demand that future protocols be designed, analyzed, and validated with care.

Formal methods offer methodologies for rigorous specification and verification of network protocols. Yet, current tools require substantial expertise to use, particularly for complex protocols. To date, most successful analysis efforts have been carried out using domain-specific tools and often by the developers of those tools. There is no single point of entry for understanding what tools are available, what inputs and expertise are required to use them, and what their capabilities and limitations are. Consequently, it is understandable why standards bodies, such as the IETF and IEEE, do not routinely use formal methods to support their work.

This article analyzes the gaps between the formal methods and networking communities. We first describe the differing views on specifications held in these communities; then we present examples where formal methods have been successfully used to improve real-world protocols, demonstrating how formal specifications and associated tools can provide a valuable design aid; and finally, we propose ideas for bridging the gaps in the future.

On Specifications

The word “*specification*” means different things to different groups. The groups we focus on here are the networking community and the formal methods (FM) community.

Specifications in the networking community. Today, networking protocols are defined by standards bodies such as the IETF, IEEE, and 3GPP. The “specifications” they develop usually consist of prose documents, annotated with pseudocode, header descriptions, state machines, message sequence charts, and so on. Often the specifications are accompanied by a *reference implementation*, which can be used to test for *interoperability*. When flaws or inconsistencies are detected during testing, the specification and/or reference imple-



The sheer number of emerging technologies and the potential risk to critical infrastructure and human lives demand that future protocols be designed, analyzed, and validated with care.



mentation is fixed. In practice, reference implementations are updated more often than specifications, so they often serve as de facto specifications of many protocols.

As an example, consider TCP, which is described in IETF RFC 793,²⁷ originally published in 1981. A state machine diagram on page 22 of the RFC shows an ACK message being sent in response to a SYN message when in the SYN-SENT state. Meanwhile, in examples on page 31, both SYN and ACK are shown as the response, rather than just ACK. A later update in RFC 1122,¹¹ published in 1989, states:

“the arrow from SYN-SENT to SYN-RCVD should be labeled with “snd SYN,ACK”, to agree with [...].”

That is, for several years, TCP implementations used a specification that was consistent with an example rather than the protocol’s state-machine diagram given in the RFC. Over time, TCP has continued to be the subject of numerous RFCs—for example, RFC 9293,¹⁴ published in 2022, supposedly corrects all the inconsistencies and bugs discovered in more than four decades of use of TCP.

Of course, RFCs and their accompanying social processes and reference implementations have been extremely successful—they are the key artifacts that have driven the development and evolution of Internet protocols over many decades. Yet, social processes and RFCs often leave ambiguities and gaps in protocol descriptions. These ambiguities and gaps can, in turn, give rise to serious security vulnerabilities, as illustrated by some later examples. The application of formal reasoning can help to discover and resolve such issues. But formal reasoning requires precise definitions of protocols and requirements, both of which are typically unavailable.

IEEE standards, issued by the various working groups under the IEEE Standards Association, are also often used as specifications. Unlike RFCs, which are at most a few hundred pages and coordinated by small groups, IEEE standards are often long and complex—for example, the 802.11

standard for Wi-Fi was developed by a group with more than 300 members and runs to over 3,500 pages. In many standards, aspects are deliberately left vague or deferred to implementations. This is also the case with IETF RFCs, since looser standards generally provide more room for innovation in implementations.

Last but not least are the “specifications” of the cellular protocols and architectures standardized by the 3rd Generation Partnership Project (3GPP), which is itself an association of seven telecommunications standards development organizations: Association of Radio Industries and Businesses (ARIB), Alliance for Telecommunications Industry Solutions (ATIS), China Communications Standards Association (CCSA), European Telecommunications Standards Institute (ETSI), Telecommunications Standards Development Society, India (TSDSI), Telecommunications Technology Association (TTA), and the Telecommunication Technology Center (TTC). 3GPP standards cover all aspects of cellular telecommunications technologies, including radio access, core network, and service capabilities. The 3GPP specifications also provide hooks for non-radio access to the core network as well as for interworking with non-3GPP networks. As with RFCs and IEEE standards, 3GPP specifications are often intentionally vague and usually do not include reference implementations. While open source cellular platforms do exist, it is usually difficult to obtain high-level specifications from code.

Specifications in the formal methods community. In the FM community, a specification is usually understood as an unambiguous description of a system and of its desired properties, both with a precise semantics. While the construction of formal specifications for protocols requires effort, this effort often pays off by providing benefits such as:

- The process of formalization brings clarity and understanding to protocol design. It is common for gaps and inconsistencies to be revealed in the process of constructing a formal model.

- A protocol works only under as-

sumptions about its environment, which may include interactions with other protocols. It is common for the process of formalization to lead to a clear understanding of these interactions and to bring hidden assumptions out into the open.

- A specification (even a partial one) can be used to automatically generate tests, which often reveal protocol errors that might not be detected by traditional interoperability testing of implementations.

- A specification may also be used to comprehensively analyze protocol properties, through formal proofs which show that desired protocol properties hold for *all* scenarios. Such proofs are especially valuable when the properties go beyond functional correctness and capture security, performance, fault tolerance, and so on.

Many associate formal methods solely with verification. As C.A.R. Hoare once said, “A program that has not been specified cannot be incorrect, it can only be surprising.” Unfortunately, the complete set of properties a protocol is expected to satisfy (also called “intents” or “requirements”), including security and performance properties as well as environmental assumptions, is rarely stated explicitly, making complete formal verification effectively impossible. What is possible is to analyze certain aspects of implementations, although there are still significant challenges. For example, an implementation may deviate from the intent of the protocol designer, or the analysis may require having a specification for the environment. Even so, validating implementations using partial specifications and lightweight FM approaches that only model certain aspects of the environment is often useful.

In principle, formal specifications of networking protocols could be used to prove certain requirements, including relating high-level specifications to low-level implementations. Formal specifications can point to bugs in the implementation as well as inconsistencies or disagreements on what the protocol “is.” When either a protocol or its implementations need to be changed, specifications can help guide refactoring and thus

increase agility. In general, formal specifications can be a valuable aid for the social processes used to design protocols.

While formal specifications have many potential benefits, there is a long way to go to make them useful and usable by the networking community. Today, applying FM to real-world protocols requires considerable skill, as well as time and money. For networking researchers it requires specifying protocols in a completely new way, which will likely slow down certain parts of the design process. There are also social issues of network developers resisting change in common practices. Later in this article, we identify barriers that prevent FM from being widely used by the designers of network protocols. But first, we describe a few examples where FM has been applied successfully in networking.

Examples of Success Stories

There have been several attempts by the FM community to capture RFCs and reference implementations as formal specifications. These attempts resulted in many successes, yet they all required significant effort and have not produced a general methodology for reverse engineering intents from documentation or code. This is hardly surprising since, as stated above, RFCs are ambiguous, unclear, and sometimes contradictory. An individual implementation, of course, is not ambiguous. Yet, it reflects design decisions that, at best, restrict the intent of the RFC and at worst create vulnerabilities and errors.

We present six examples here. The first two concern subtle security and privacy properties and were carried out by expert teams. The next three examples use lightweight FM for modeling and testing. These methods can, with further development by the FM community, be adopted by practitioners. If adopted, it is highly likely they would result in increased productivity for protocol designers. The last example discusses the use of FM in software-defined networks (SDNs), including work that has connected formal specifications to real-world implementations. The table summarizes the tools discussed in

the examples.

Transport layer security. One of the success stories in formally analyzing a widely used protocol is the formal analysis conducted on both versions 1.2 and 1.3 of the transport layer security (TLS) protocol. TLS is the main means for securing communications on the Internet. It was originally created by Netscape in 1995 under the name SSL and released as open source under the name TLS 1.0. The protocol went through many changes, with version 1.2 being used for many years. Significant effort went into formally analyzing TLS, as discussed below.

In 2013, Bhargavan et al.⁹ developed a verified reference implementation of TLS 1.2, which follows the RFC specification.²⁹ TLS is a very complex protocol with numerous configuration and deployment options, supporting many ciphers and backward compatibility. The authors had to make decisions about what to support, given that some configurations and some supported ciphers are known to be broken. Their reference implementation included the two main components: the authenticated stream encryption for the re-

cord layer and key establishment for the handshake. The verification was done using F*. The authors did not formally account for side-channel attacks based on timing and acknowledge that proving the absence of such attacks would require different tools.

The design of TLS 1.3 represented a big change, and 28 drafts were created until the design was finalized. One of the biggest changes was reducing the latency of the handshake protocol and supporting 0-RTT connection resumption, as well as by default supporting perfect forward secrecy. A modular symbolic model of the TLS 1.3-Draft 21 release candidate, with TAMARIN verification of the security requirements claimed in the draft, are in Cremers et al.¹²

The focus of Cremers et al.¹² is the handshake protocol, as it was essential for the overall security of the communication. The analysis revealed an undesired behavior that had an impact on the strong authentication guarantees in some implementations of the protocol. As this work was conducted during the design process of TLS 1.3, it had a strong influence on the final draft.

Around the same time, Bhargavan et al. presented verified symbolic and computational models of TLS 1.3-Draft 18.⁸ The models were specified using ProVerif and evaluated considering known and previously unpublished vulnerabilities that were considered during the design of TLS 1.3. This work presents the first machine-checked cryptographic proof for a CryptoVerif model of TLS 1.3 Draft-18.

With respect to the TLS 1.3 finalized RFC, the most recent effort¹³ developed and verified a reference implementation of the TLS 1.3 Record Layer protocol and its cryptographic algorithms. The model was specified and verified using F*.

5G Authentication and Key Agreement. Another important standard is 5G, the current architecture and set of protocols for cellular networks. A key component of the standard is the 5G Authentication and Key Agreement, or 5G-AKA for short, which describes how a *user device* and a customer's *home network* agree on a shared key. This protocol is critical for the security of communication on 5G networks, as other keys are derived from this shared key, which are in turn used to protect users' data security, the authenticity of messages and calls they receive, the connections they start, and even billing based on usage.

The work in Basin et al.⁵ reports on a formal analysis of the 5G-AKA using TAMARIN. The protocol was formalized based on 3GPP's TS 33.501 document, and the authors had additional access to 5G specialists. The formalization was challenging given 5G-AKA's complexity, both due to the specification's size and its usage in different contexts. For example, when roaming, a user's device may connect to mobile networks (called serving networks) that are different from the service provider. The protocol then connects three parties rather than just two, where only two of the parties—the user's device and home network—share initial secrets. Other complexities arise due to technological constraints and the need for backward compatibility.

The formal analysis started with an in-depth study of the relevant protocol documents, as well as discus-

Table. Summary of verification tools mentioned in examples.

Tool	Main Usage	Website
Alloy	A widely used language and analyzer for software modeling.	https://alloytools.org
CryptoVerif	A protocol prover that provides for specifying the security assumptions on cryptographic primitives.	https://opam.ocaml.org/packages/cryptoverif/
F*	A general-purpose, proof-oriented programming language that combines dependent types with proof automation based on SMT solving and tactic-based interactive theorem proving.	https://fstar-lang.org
ProVerif	An automatic cryptographic protocol verifier in the Dolev-Yao model, based on representing protocols by Horn clauses.	https://bblanche.gitlabpages.inria.fr/proverif/
IVy	A language and a tool for specifying, modeling, implementing, and verifying distributed protocols.	https://microsoft.github.io/ivy/language.html
Rocq (Coq)	A proof assistant for reasoning about software correctness.	https://coq.inria.fr
SPIN	A tool for verifying temporal properties of software models.	https://spinroot.com/spin/whatispin.html
TAMARIN	A tool for analyzing cryptographic protocols using symbolic cryptography model.	https://tamarin-prover.com
Z3	A widely used automated theorem prover and constraint solver for symbolic logic.	https://github.com/Z3Prover/z3

sions with some experts involved in the standardization. Afterward, an abstract version of the protocol was formalized in TAMARIN. The majority of the effort was the several person-months needed to understand the specification. In contrast, the time needed to subsequently formalize the resulting model was relatively short. Some additional person-months were needed to verify the protocol's security properties, in particular writing proof strategies to help automate the construction of TAMARIN proofs.

During verification, several flaws were found, which were reported to the 3GPP. The flaws were subsequently fixed in the standard, with one exception. The exception concerned the privacy of the user's identity, which may be violated in the 5G-AKA protocol with a fairly simple replay attack that exploits the 5G-AKA's resynchronization protocol with certain counters. This problem cannot be easily fixed in the current design, but a further iteration could solve this problem by shifting away from counter-based mechanisms.

The most critical vulnerability discovered by this effort was a protocol error that allowed the attacker to induce confusion between users for the home network—that is, the date or time that is used and should be billed to customer *A* would be incorrectly billed to another customer *B*. The vulnerability was disclosed and fixed. As a result, this verification work was admitted to the “GSMA Mobile Security Research Hall of Fame” as CVD #0012, 2018.

Quick UDP Internet Connection. Quick UDP Internet Connection (QUIC) is a new Internet secure transport protocol that has gone through more than six years of IETF standardization; its 35th version was declared as the standard in 2022 (RFC 9000). QUIC is intended as a replacement for the TLS/TCP stack and will be the basis of HTTP/3, the next official version of the hypertext transfer protocol. It already carries a substantial amount of Internet traffic and is expected to carry even more in the future.

QUIC's pivotal role rendered it a good candidate for the methodology of *network-centric compositional testing* (NCT), introduced by McMillan



The compositional nature of the NCT approach enables detecting cases when the specification is either too weak or too strong.



and Zuck.^{22,23} The term “*network-centric*” refers to describing a protocol's behavior as observed on the wire, rather than its abstract implementation. This approach is compositional, in the sense that any assumption made on the input of one process (or component) in the protocol can be treated as a requirement on the output of other processes (or components). *Compositionality* allows one to infer from the fact that implementations pass all tests they will interoperate correctly when composed. It also enables the discovery of cases where the specification is too strong (rejecting legal protocol behaviors) or too weak (peers misbehave when receiving an input the specification allows).

The work reported in McMillan and Zuck^{22,23} developed formal wire specifications for parts of QUIC and tested implementations for compliance using these specifications. The testers take on one of the roles of the protocol. They generate packets that are legal according to the wire specification but may or may not be produced by any existing implementation. This generation can be done, for example, using a modified SMT solver that randomly solves the constraints occurring in specifications, in such a way that every legal packet at a given point in the execution of a protocol has a non-zero probability of being generated.

This approach to testing has several advantages. It makes it possible to resolve ambiguities in informal standards documents using knowledge inherent in the implementations. At the same time, it verifies that the implementations comply with the formal specification as it is developed. It also exposes the implementations of a given protocol role to legal behaviors of the other role that may not be produced by existing implementations but may be produced by future implementations. Thus, it can detect interoperability issues before they occur in the wild.

Finally, the compositional nature of the NCT approach enables detecting cases when the specification is either too weak or too strong, allowing the developer to refine the specification accordingly. Consider a protocol with two roles *X* and *Y*. When testing

an implementation of role X , the output of role Y from the specification of role Y is generated. If a response by the implementation of X is not allowed by the specification of role X , then either the implementation is not compliant, or the specification of X is too restrictive. On the other hand, if a response is not accepted by the implementation of X , then either the implementation of X is not compliant, or the specification of Y that allowed the response is too permissive.

The process of formally specifying QUIC uncovered numerous errors in the reference implementations, as well as some issues in the the draft standards themselves. Examples of such issues include off-path denial of service attacks, one of which was a result of the standard as it appeared then in a draft RFC, and an information leak similar to the “heartbleed” vulnerability in OpenSSL.

NCT was demonstrated on QUIC, but the method is quite general and has also been applied to simpler protocols in classroom settings. The experiments used the IVy tool, which in turn relies on the Z3 theorem prover. The size and complexity of QUIC packets presented challenges for efficient automated test generation. To overcome this problem, the specification must be carefully structured, a task that requires expertise on the part of the user.²²

NCT has several features that may make it applicable to other complex protocols:

- It allows testing of *partial* specifications—that is, one need not specify the whole protocol to enjoy the benefits of testing the parts captured in the formal model.
- The random nature of the generated tests allows testing of corner cases that interoperability testing rarely reveals.
- The randomized tests, while not originally intended as such, turned out to be useful as adversarial inputs, thus allowing detection of security flaws. Applying NCT to QUIC revealed that many security flaws can be detected using randomized testing, similar to how fuzzing has been effective for finding vulnerabilities in general-purpose software.

Chord. The Chord distributed hash

The process of formally specifying QUIC uncovered numerous errors in the reference implementations, as well as some issues in the the draft standards themselves.

table was first presented in an ACM SIGCOMM paper that garnered its authors the 2011 Test-of-Time Award.¹⁸ An RFC based on Chord has also been developed.²¹ A Chord instance is a ring-shaped peer-to-peer network, and the papers specify the protocol for maintaining the ring in pseudo-code and text. Correctness of this protocol is a form of eventual consistency, meaning the protocol can eventually repair all defects due to failures, so that all live members can continue to reach each other. The SIGCOMM paper states, “Three features that distinguish Chord from many other peer-to-peer lookup protocols are its simplicity, provable correctness, and provable performance.”¹⁸

However, despite these assurances, the work in Zave³⁷ shows that, under the same assumptions made in the Chord papers, the ring-maintenance protocol is not correct, and not one of the seven properties claimed as invariants actually holds. These results were obtained through automated checking of a simple formal model in the Alloy language. Employees at Amazon Web Services (AWS) credit this work for motivating them to start using formal methods.²⁴

Most of the seven claimed invariants are obviously useful and important, either for ring maintenance or for key lookup. By fixing the flaws revealed by formal modeling and analysis, Zave³⁸ shows it is possible to specify the Chord protocol in a way that is both correct and as efficient as the original. The proof of correctness is particularly interesting, because although the specified protocol satisfies the claimed properties, the properties do not resemble the true invariant upon which the proof is based.

Session Initiation Protocol (SIP). SIP is the dominant signaling protocol for IP-based audio, video, and multimedia communication, and has been standardized by the IETF. The baseline SIP protocol is defined in a 268-page informal document.³¹ Due to extensions, interoperation with other protocols, and explanatory material, its description as of 2009 consisted of 142 documents totaling tens of thousands of pages.³⁰

With a complex protocol specification such as this, it is not surprising

that people working with SIP spend many hours trying to answer basic questions such as, “Can a protocol endpoint in state S send a message of type m ?” In this context, Zave³⁶ provides a simple formal model in Promela (the language of the model-checker SPIN) that proved to be an invaluable resource for software developers. Model checking with SPIN verified that the model does not deadlock, that it is complete in the sense of including every message that can be received in every state, and that all parts of the model are reachable. The model provides quick and reliable answers to programming questions. It has also been used in Bond et al.¹⁰ to generate automatically a large number of test cases. In addition to these fundamental uses, the model answered several longstanding questions, as discussed below.

Consistency. Regardless of how it is described, a protocol should be internally consistent. SIP experts worry that SIP’s many extensions have introduced inconsistencies, meaning that a new behavior in an extension violates an assertion or constraint in an existing document. These experts are aware that inconsistencies could easily survive the documentation and standardization process and, in fact, their fears are well founded. Even two of the earliest extensions were shown to violate fundamental assumptions of the protocol.³⁶

Complexity. A widely used protocol should not be unnecessarily complex. Each capability should be generalized as much as is reasonable and convenient, in preference to adding new capabilities that accomplish similar and overlapping goals. The most important piece of SIP is the *invite* transaction, a three-way handshake allowing two endpoints to set up and negotiate the parameters of a set of media channels between them. Two early extensions serve the same function as the *invite* transaction, in different but overlapping circumstances. The cost of these extensions is considerable. They require five new message types, and a simplified model of just one of them requires 11 states and 15 state transitions.³⁶ Yet, with the addition of a single Boolean flag to the messages of the *invite* transaction, all

this additional complexity could be avoided, and all media-control functions could be performed with *invite* transactions alone.

Race conditions. A race condition occurs when two messages cross in transit, either going the same direction (Figure 1 left) or different directions (Figure 1 right). Either kind of race can happen in SIP, especially when SIP messages are sent via UDP instead of TCP at the transport level (either is allowed).

Many SIP documents use message-sequence charts to show particular common scenarios. These charts are rendered in ASCII by means of IETF macros, and look like Figure 2 (Note that these charts represent message transmission as instantaneous so that race conditions are impossible). Not surprisingly, SIP race conditions are

not well documented, and their handling is incompletely standardized.

A SIP document¹⁶ many years in the making records seven possible race conditions within the scope of the Promela model.³⁶ The Promela model points to race conditions wherever they show that a message can be received when it is not expected or desired. By examining these places in the model, it is straightforward to find the same seven race conditions and 42 others. Note that later IETF documents, such as Hasebe et al.,¹⁶ show that ASCII art for race conditions has been developed. The first specification of proxy (transparent) behavior for SIP was also based on the formal model.¹⁰

Software-defined networks. Software-defined networks and programmable networks more generally make

Figure 1. Race conditions arise when messages going in the same or opposite directions cross in transit.

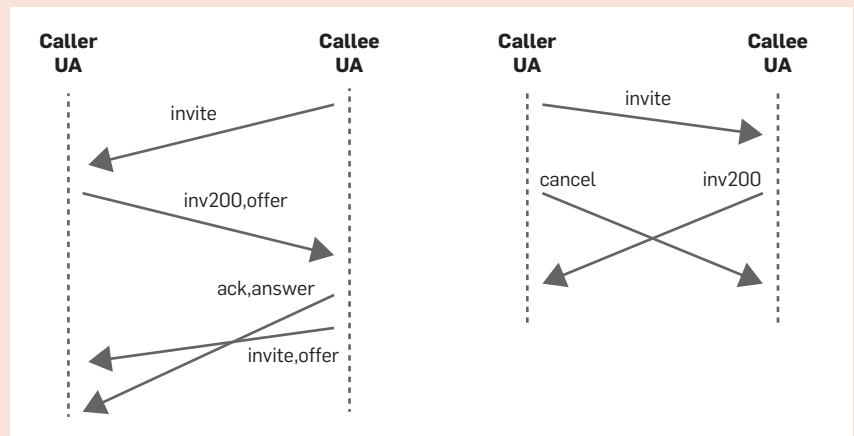
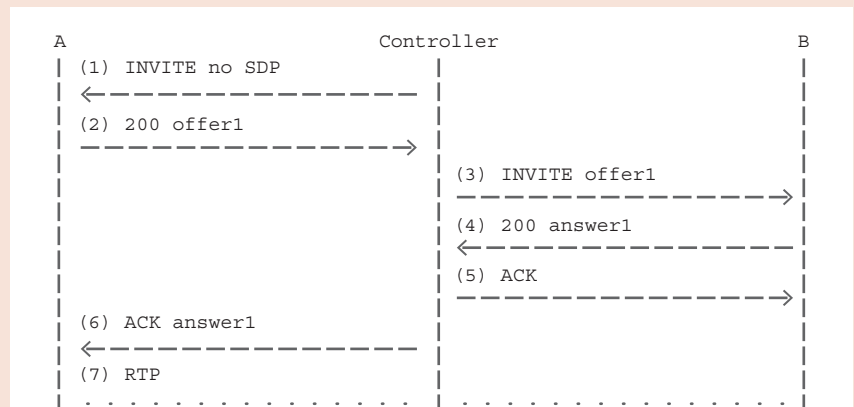


Figure 2. Because the IETF standard for message-sequence charts shows instantaneous message delivery, race conditions appear to be impossible.



it possible to formally specify and verify the *low-level implementations* of network infrastructure. Given the number of interacting systems, the complexity of the software in network components, and the need for decentralized control, making network infrastructure programmable without also supplying “power tools” from FM is inherently risky. Thus, even the first instances of programmable networking (for example, SwitchWare^{1,2} and Active Bridge³) were designed to eliminate large classes of errors by using strongly typed programming languages, such as Standard ML and OCaml.

The emergence of SDN, which opened up interfaces for programming the network infrastructure, also led to significant interest in applying FM to network infrastructure. Building on early work that used static analysis to analyze router configurations,³⁴ tools such as Veriflow,¹⁹ Header Space Analysis,¹⁸ NetKAT,⁴ and Atomic Predicates³⁵ allow operators to specify and verify network-wide data plane properties, such as connectivity, loop freedom, way-pointing, and so on. These early tools often worked with abstract models of network devices, but a number of recent tools have closed the gap with lower-level hardware models^{20,32} and also explored applications of runtime verification.²⁸ A complementary line of research extended work on network verification to handle richer models of network functions, such as load balancers, firewalls, content caches, and so on.²⁶ The distinguishing feature of these network functions is they are written in general-purpose languages like C/C++ and typically rely on state—for example, to track flows. Yet another line of work developed techniques for reasoning about conventional control-plane protocols, such as BGP and OSPF.¹⁵

The Gap

The community that develops Internet protocols aims toward correct and secure protocols. Current descriptions of the protocols do not allow for analysis, and thus for guarantees of correctness or security. FM is the most promising means for achieving these goals, but it requires formal specifications. Hence, there is

a gap between the ways protocols are currently defined and the potential of FM to improve protocols throughout their lifecycles.

The purpose of this section is to point out specific challenges that must be addressed to fill the gap. But first, we would like to note two issues often thought of as obstacles to the use of FM, but are not:

1. *Incompleteness*: Protocol specifications are usually incomplete, often for good reasons: Some details are best left to hardware and software choices in implementations, vendors want space to compete, there are extensions to the original protocol, or there are simply issues that were not well understood when the specification was written. Because the essence of FM is abstraction, where “abstraction” is another way of saying “incompleteness,” formal specification languages are actually far better than implementation-oriented languages for this purpose. A well-chosen specification language enables its users to state essential facts while being perfectly clear about what it is *not* saying.

2. *Scale*: In the same vein, scalability and agility are not barriers to obtaining the practical benefits of formal specification in networking. Though informal specification documents are often voluminous and complex, we have found in practice that a small number of formal methods experts can effectively capture the formalizations of complex protocols, such as QUIC, and keep up with changes as a protocol evolves. Moreover, existing tools have sufficient capacity to handle these specifications. There are several reasons for this. First, the specification need not include every detail, as noted above. Second, and more importantly, these efforts do not involve detailed proofs about code or hardware. Rather, they are applications of *lightweight* FM. They use relatively inexpensive techniques, such as specification-based testing, to resolve ambiguities in informal documents and reserve proof for simple abstract models. This makes it possible to obtain the benefits of formalization we have outlined at a moderate cost in human and computational effort, albeit by sacrificing the ambitious goal of certain

correctness. Nevertheless, significant challenges remain; we have identified the following:

Semantic divide: The FM community wishes for specifications that are rigorous and unambiguous. Such specifications serve as thinking tools, contracts between protocols and their environments (or even different parts of a protocol), as well as the basis for formal proofs, rigorous testing, and automated code extraction and synthesis.

The community specifying Internet protocols wishes for consensus among stakeholders, expediency, and flexibility for vendors who need to distinguish their products. This implies that the specifications of the protocols should be easy to read and understand. Practitioners are more comfortable with natural language, despite its ambiguities, than with formal specifications.

Finding the right tool: Navigating among the multitude of tools available, and finding those that may be helpful, is challenging. The examples we provided highlight the problem: Several different tools were used for modeling the protocols, and the analysis was conducted by researchers who either created the tool they used or had significant experience with it. The lack of commonality among tools impedes their developers as well. For examples of a different approach, observe how agreement on the DIMACS SAT format spurred the competition to develop better SAT tools, and similarly for SMTlib. Similarly, Rocq (previously Coq) and Lean are becoming de facto standards, enabling the reuse of libraries of proofs across projects.

Using the tools: Even when practitioners find a suitable formal language and tool, and succeed in encoding their protocols as formal specifications, they find that the supporting tools require a considerable amount of expertise and training to use. Even with practice, the tools may be much more difficult to use than necessary.


This is particularly unfortunate because good tools can do so much. They can make specifications more comprehensible with visualizations and alternative views. They allow analysis and verification, they can

simulate behavior and generate test cases, along with other functions we have mentioned. Although it is always important to compare the formal semantics of a specification to the assumptions and expectations in its users' minds—the step called *validation*—experience shows that a variety of validation checks can be assisted with automated tools.³⁹


Limited support for quantitative properties: Historically, much of the focus of the FM community has been on qualitative aspects of system behavior—for example, functional correctness. However, important properties of network protocols are often quantitative or even statistical in nature, and there are very few tools to handle such properties. Protocols are expected to behave “well” most of the time and be reactive to ever-changing environmental conditions. There are very few formal frameworks, and consequently, tools, that can deal with such complex properties.

Limited support for security properties: Security adds another dimension to the gap as security properties are inherently difficult to formalize and check. Some require reasoning about “hyperproperties” (properties that involve several runs of the protocol), while others require complex cryptographic analyses. All require reasoning about unknown adversaries including side-channels.

Research community gap: Experiments performed by Gerard Holzmann at Bell Labs in the 1980s¹⁶ demonstrated that systems developed using FM take less time to construct than systems built using a “develop first, analyze later” approach. While Holzmann’s study only provides a single data point, recent work using FM in the design of cloud services and cryptographic hardware modules hints at the potential advantages of integrating FM into the design process. Even so, it seems difficult to convince most developers that designing a system carefully and formally may actually save them time. The prevailing wisdom appears to be that just writing something up that (kind of) works and then debugging, testing, and reaching a social consensus that glosses over details and (kind of) cap-



The community specifying Internet protocols wishes for consensus among stakeholders, expediency, and flexibility for vendors who need to distinguish their products.



tures what has been implemented seems faster.

The IETF (and 3GPP) develop their protocols through an iterative debugging process. This approach is unlikely to change. (The process also considers non-correctness factors such as messaging overhead, expected protocol convergence time, and support for multiple features.) It is the responsibility of the research community to create methodologies and tools that will convince designers and developers to use them while designing a protocol. Likewise, it is the role of the FM community to develop techniques that match the iterative development of networking protocols, and demonstrate, as in Holzmann’s work, that formal specifications can considerably speed it up.

How to Bridge the Gap

Given the large gap between the potential benefits of FM and the ability of protocol designers to use existing tools, we discuss several directions that can narrow this gap.

1. *Engage industry.* One way forward is to demonstrate the benefits of FM in industry, by finding bugs that really matter. This was the case, for example, with work on the EMV payment protocol using TAMARIN.^{6,7} After finding serious flaws that allowed researchers to carry out high-value transactions without a PIN (the second authentication factor required for such transactions), the researchers began a collaboration with one of the EMV vendors that lead to a new EMV kernel (subprotocol), C-8, which was formally verified.

2. *Develop tools that are easy to use.* The FM community needs to work on building usable and useful tools. Whether a tool is “usable” or “useful” depends on the audience the tool is meant to serve. Hence, more collaborations between the two communities are needed, as well as with human-factors researchers.

3. *Develop tools that address specific needs in the networking community.* One of the problems of formal studies of networking protocols is the lack of a good model of the environment as well as precise requirements from the protocols that operate in these environments. For example, consider

congestion control protocols, which have received significant attention in the networking community. Without a good model of the environment, it is virtually impossible to identify which of the many congestion control protocols should be used and when. Even for simple Additive-Increase/Multiplicative-Decrease (AIMD) protocols, such as TCP New Reno, a theoretical study identified pathological cases where the protocol gets stuck in its slow-start state.⁴⁰ Given initial parameters, lightweight formal tools can help identify conditions on the environment that avoid this undesirable behavior.

4. *Integrate design and verification.* The design process used for TLS 1.3 illustrated the benefits of performing formal analysis on intermediate drafts and not only on the final product. However, incorporating FM into the development process requires a considerable investment of time and currently requires experts to perform the analysis. This cost can be decreased by better tools that can be easily used by non-experts and should be amortized over the long term, as fewer bugs and vulnerabilities will be present. Additional automated tools facilitating collaborations, detecting ambiguities and inconsistencies in textual specifications, and tracking changes are needed (and some of them are already being used) to ensure end-to-end integration of design and verification. Bringing together networking and formal-methods researchers is essential in identifying the needs for such tools and ensuring cooperation in changing the design process.

5. *Develop formal notations to express protocols and an ecosystem around them.* There are at least two major research opportunities to address the semantic divide. One is the development of a standard formal protocol notation (or notations). It is typical for a verification tool to define its own distinct input language, tailored to a class of protocols or to a type of verification method. As a result, models developed with different tools cannot be shared or interlinked, making it difficult to apply a combination of verification methods, as is often required for a



Incorporating FM into the development process requires a considerable investment of time and currently requires experts to perform the analysis.



complete protocol analysis. The second opportunity is in exploiting the recent remarkable advances in natural language processing (NLP), such as large language models (LLMs), to automatically translate human-readable protocol specifications to a machine-analyzable formal notation.²⁵ A key challenge is to find ways to cope with the gaps, errors, and ambiguities that may be present in protocol specifications, while also coping with the artificial errors (for example, hallucinations) that could be introduced during the translation. Success in addressing these challenges would have a revolutionary impact on improving the quality of protocol specifications.

DARPA's Open Programmable Secure 5G (OPS-5G) program developed techniques for automatically translating standards documents into precise executable models using NLP. The highly structured nature of standards documents and their limited domain of discourse make them ideal targets of opportunity for NLP. Disagreements within a specification result in security holes, as developers cope with differences in interpretation. Having an automated system for extracting an executable reference implementation from an informal specification provides a way to validate proprietary code developed by equipment vendors.

6. *Continue to integrate FM in computer science education.* FM has been increasingly recognized and integrated into computer science (CS) education. Many CS programs have started integrating FM into their curricula, either as standalone courses or as components of existing courses such as software engineering, algorithms, or programming languages. Despite the progress, there are still challenges in teaching FM effectively. These challenges include the abstract nature of formal techniques, the steep learning curve for some concepts, and the need for specialized tools and expertise.


7. *Continue to increase awareness about the benefits of applying formal methods for network protocols.* We need to increase awareness of the benefits of existing formal analysis and tools in the networking community and beyond. For one, it seems the networking community is not fully

aware of the full benefits of FM. Perhaps methods that can highlight the potential benefits—not more examples to solve—can help to raise awareness. While summer schools were organized in the past, they focused on formal methods more generally. Perhaps a summer school or a Dagstuhl seminar with a focus on network protocols bringing together members of the two communities would help in this regard.

Conclusion

Formal methods offer methodologies and tools to rigorously specify protocols, verify parts of them, and even test their specifications and implementations. Numerous success stories demonstrate that these benefits can be achieved in practice, reducing the costs associated with fixing bugs and vulnerabilities. However, several obstacles prevent their wider adoption and use by non-experts. In this article, we identified the gaps that prevent this adoption and described directions that can be taken to narrow the gap.

Acknowledgments

The authors would like to thank the anonymous reviewers for many helpful comments and suggestions for improvements. David Basin thanks the Werner Siemens-Stiftung for their generous support of this project, under the Centre for Cyber Trust. Nate Foster was supported in part under NSF grant SHF-1918396, ONR contract N6833522C0411, DARPA contract HR001124C0429, and gifts from Juniper, Google, and the VMware University Research Fund. Kedar Namjoshi was supported in part by DARPA under contract HR001120C0159. Cristina Nita-Rotaru and Lenore Zuck thank the NSF in suggesting and encouraging this work, which was partially funded with grant CNS-2140207. The work of Lenore Zuck was also funded by the Discovery Partners Institute of the University of Illinois grant 634024 and by the NSF grant SHF-1918429. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of any of the funding institutions and companies. 

References

- Alexander, D.S. et al. The SwitchWare active network architecture. *IEEE Network Magazine: Active and Programmable Networks* 12, 3 (1998), 29–36.
- Alexander, D.S., Arbaugh, W.A., Keromytis, A.D., and Smith, J.M. A secure active network environment architecture: Realization in SwitchWare. *IEEE Network Magazine: Active and Programmable Networks* 12, 3 (1998), 37–45.
- Alexander, D.S., Shaw, M., Nettles, S.M., and Smith, J.M. Active bridging. In *Proceedings of the ACM SIGCOMM Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication* (1997), 101–111.
- Anderson, C.J. et al. NetKAT: Semantic foundations for networks. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages* (2014), 113–126.
- Basin, D. et al. A formal analysis of 5G authentication. In *Proceedings of the 2018 ACM SIGSAC Conf. on Computer and Communications Security* (2018), 1383–1396.
- Basin, D.A., Sasse, R., and Toro-Pozo, J. Card brand mixup attack: Bypassing the PIN in non-visa cards by using them for Visa transactions. In *Proceedings of the 30th USENIX Security Symp.* (2021), 179–194.
- Basin, D.A., Sasse, R., and Toro-Pozo, J. The EMV standard: Break, fix, verify. In *Proceedings of the IEEE Symp. Security and Privacy* (2021), 1766–1781.
- Bhargavan, K., Blanchet, B., and Kobeissi, N. Verified models and reference implementations for the TLS 1.3 standard candidate. In *Proceedings of the IEEE Symp. Security and Privacy* (2017), 483–502.
- Bhargavan, K. et al. Implementing TLS with verified cryptographic security. In *Proceedings of the IEEE Symp. Security and Privacy* (2013), 445–459.
- Bond, G.W., Cheung, E., Smith, T.M., and Zave, P. Specification and evaluation of transparent behavior for SIP back-to-back user agents. In *Proceedings of IPTComm* (2010), 48–58.
- Braden, R.T. Requirements for Internet Hosts—Communication Layers. RFC 1122, (Oct. 1989).
- Cremers, C. et al. A comprehensive symbolic analysis of TLS 1.3. In *Proceedings of the 2017 ACM SIGSAC Conf. on Computer and Communications Security*, 1773–1788.
- Delignat-Lavaud, A. et al. Implementing and proving the TLS 1.3 record layer. In *Proceedings of the IEEE Security and Privacy Symp.* (2017), 463–482.
- Eddy, W. Transmission Control Protocol (TCP). RFC 9293, (Aug. 2022).
- Fogel, A. et al. A general approach to network configuration analysis. In *Proceedings of the USENIX Conf. NSDI* (2015), 469–483.
- Hasebe, M. et al. Example call flows of race conditions in the Session Initiation Protocol (SIP). RFC 5407, Dec. 2008.
- Holzmann, G.J. The theory and practice of a formal method: NewCoRe. In *Proceedings of the IFIP World Computer Congress* (1994), 35–44.
- Kazemian, P., Varghese, G., and McKeown, N. Header space analysis: Static checking for networks. In *Proceedings of the USENIX NSDI Symp.* (2012), 113–126.
- Khurshid, A. et al. VeriFlow: Verifying Network-Wide invariants in real time. In *Proceedings of the USENIX NSDI Symp.* (2013), 15–27.
- Liu, J. et al. p4v: practical verification for programmable data planes. In *Proceedings of the ACM SIGCOMM Conf.* (2018), 490–503.
- Maenpaa, J. and Camarillo, G. Self-tuning distributed hash table (DHT) for REsource LOcation And Discovery (RELOAD). RFC 7363, (Sept. 2014).
- McMillan, K.L. and Zuck, L.D. Compositional testing of Internet protocols. In *Proceedings of IEEE Cybersecurity Development Conf.* (2019), 161–174.
- McMillan, K.L. and Zuck, L.D. Formal specification and testing of QUIC. In *Proceedings of ACM SIGCOMM* (2019), 227–240.
- Newcombe, C. et al. How Amazon Web Services uses formal methods. *Communications of the ACM* 58, 4 (Apr. 2015), 66–73.
- Pacheco, M.L. et al. Automated attack synthesis by extracting finite state machines from protocol specification documents. In *Proceedings of the IEEE Security and Privacy Symp.* (2022), 51–68.
- Pirelli, S., Valentukonyte, A., Argyraki, K., and Candea, G. Automated verification of network function binaries. In *Proceedings of the USENIX NSDI Symp.* (2022), 585–600.
- Postel, J. Transmission Control Protocol. RFC 793, Sept. 1981.
- Renganathan, S. et al. Hydra: Effective runtime network verification. In *Proceedings of the ACM SIGCOMM Conf.* (2023), 182–194.
- Rescorla, E. and Dierks, T. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Aug. 2008).
- Rosenberg, J. A hitchhiker's guide to the Session Initiation Protocol (SIP). RFC 5411 (Jan. 2009).
- Rosenberg, J. et al. SIP: Session Initiation Protocol. RFC 3261, June 2002.
- Ruffy, F. et al. P4testgen: An extensible test oracle for p4. In *Proceedings of the ACM SIGCOMM Conf.* (2023), 136–151.
- Stoica, I. et al. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the ACM SIGCOMM Conf.* (2001), 149–160.
- Xie, G.G. et al. On static reachability analysis of IP networks. In *Proceedings of the IEEE Joint Conf. INFOCOMM* (2005), 2170–2183.
- Yang, H. and Lam, S.S. Real-time verification of network properties using atomic predicates. In *IEEE/ACM Trans. Networking* (2013), 1–11.
- Zave, P. Understanding SIP through model-checking. In *Proceedings of Principles, Systems, and Applications of IP Telecommunications: IPTComm 2008*, 256–279.
- Zave, P. Using lightweight modeling to understand Chord. *ACM SIGCOMM CCR* 42, 2 (Apr. 2012), 50–57.
- Zave, P. Reasoning about identifier spaces: How to make Chord correct. *ACM/IEEE Transactions on Software Engineering* 43, 12 (Dec. 2017), 1144–1156.
- Zave, P. and Nelson, T. Validation of formal models: A case study. *The Practice of Formal Methods (Essays in Honour of Cliff Jones), Part II*. Springer, (2024).
- Zuck, L.D. and Wen, Z. Avoiding spurious timeouts. In *Principles of Verification: Cycling the Probabilistic Landscape: Essays Dedicated to Joost-Pieter Katoen on the Occasion of His 60th Birthday, Part III*. Springer, 2025, 338–350.

David Basin is a professor at ETH Zurich, Zurich Switzerland.

Nate Foster is a professor at Cornell University, NY, USA.

Kenneth L. McMillan is a professor at the University of Texas, Austin, TX, USA.

Kedar S. Namjoshi is a distinguished member of Technical Staff at Nokia Bell Labs, Murray Hill, NJ, USA.

Cristina Nita-Rotaru is a professor at Northeastern University, Boston, MA, USA.

Jonathan M. Smith is a professor at the University of Pennsylvania, PA, USA.

Pamela Zave is a research specialist at Princeton University, NJ, USA.

Lenore D. Zuck is a research professor at the University of Illinois Chicago, and a lead scientist at the Discovery Partners Institute, Chicago, Illinois, USA.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.



Watch the authors discuss this work in the exclusive *Communications* video.
<https://cacm.acm.org/videos/it-takes-a-village>