Formal Verification of Standards for Distance Vector Routing Protocols

Karthikeyan Bhargavan, Davor Obradovic, and Carl A. Gunter
Department of Computer and Information Science
University of Pennsylvania

December 29, 1999

Abstract

We show how to use an interactive theorem prover, HOL, together with a model checker, SPIN, to prove key properties of distance vector routing protocols. We do three case studies: correctness of the RIP standard, a sharp realtime bound on RIP stability, and preservation of loop-freedom in AODV, a distance vector protocol for wireless networks. We develop verification techniques suited to routing protocols generally. These case studies show significant benefits from automated support in reduced verification workload and assistance in finding new insights and gaps for standard specifications.

1 Introduction

The aim of this paper is to study how methods of automated reasoning can be used to prove properties of network routing protocols. We carry out three case studies based on distance vector routing. In each such study we provide a proof that is automated and formal in the sense that a computer aided the construction and checking of the proof using formal mathematical logic. We are able to show that automated verification of key properties is feasible based on the IETF standard or draft specifications, and that efforts to achieve automated proofs can aid the discovery of useful properties and direct attention to potentially troublesome boundary cases. Automated proofs can also supplement other means of assurance like manual mathematical proofs and testing by easing the workload of tedious checking of cases and providing more thorough analyses of certain kinds of conditions.

1.1 The Case Studies

The first case study proves the correctness of the asynchronous distributed Bellman-Ford protocol as specified in the IETF RIP standard ([7, 12]). The classic proof of a 'pure' form of the protocol is given in [2]. Our result covers additional features included in the standard to improve realtime response times (*e.g.* split horizons and poison reverse). These features add additional cases to be considered in the proof, but the automated support reduces the impact of this complexity. Adding these extensions make the theory better match the standard, and hence also its implementations. Our proof also uses a different technique from the one in [2], providing some noteworthy properties about network stability.

Our second case study provides a sharp realtime convergence bound on RIP in terms of the radius of the network around its nodes. In the worst case, the Bellman-Ford protocol has a convergence time as bad as the number of nodes in the network. However, if the maximum number of hops any source needs to traverse to reach a destination is k (the radius around the destination) and there are no link changes, then RIP will converge in k timeout intervals for this destination. It is easy to see that convergence occurs within $2 \cdot (k-1)$ intervals, but the proof of the sharp bound of k is complicated by the number of cases that need to be checked: we show how to use automated support to do this verification, based on the approach developed in the previous case study. Thus, if a network has a maximum radius of 5 for each of its destinations, then it will converge in at most 5 intervals, even if the network has 100 nodes. Assuming the timing intervals in the RIP standard, such a network will converge within 15 minutes if there are no link changes. We did not find a statement of this result in the literature, but it may be folklore knowledge. Our main point is to show how automated support can cover realtime properties of routing protocols.

Our third case study is intended to explore how automated support can assist *new* protocol development efforts. We consider a distance vector routing protocol arising from work at MANET, the IETF work group for mobile ad hoc networks. The specific choice is the Ad-Hoc On-Demand Distance Vector (AODV) protocol of Perkins and Royer [18], as specified in the second version of the IETF Internet Draft [17]. This protocol uses sequence numbers to protect against the formation of loops, a widely noted shortcoming of RIP. A proof that loops cannot form is given in [18]. We show how to derive this property from a general invariant for the paths formed by AODV, essentially recasting the proof by contradiction in [18] as a positive result, and use this invariant to analyze some conditions concerning failures that are not fully specified in [17] but could affect preservation of the key invariant if not treated properly. Our primary conclusion is that the automated verification tools can aid analysis of emerging protocol specifications on acceptable scales of effort and 'time-to-market'.

1.2 Verification of Networking Standards

Automated logical reasoning about computer systems, widely known as *formal methods*, has been successful in a number of domains. Proving properties of computer instruction sets is perhaps the most established application and many of the major hardware vendors have programs to do modeling and verification of their systems using formal methods. Another area of success is safety critical devices. For instance, [6] studies invariants of a weapons control panel for submarines modeled from the contractor design documents. The study led to a good simulator for the panel and located some serious safety violations. The application of formal methods to software has been a slower process, but there has been noteworthy success with avionic systems, air traffic control systems, and others. One key impediment in applying formal methods to non-safety-critical systems concerns the existence of a specification of the software system: it is necessary to know what the software is intended to *satisfy* before a verification is possible. For many software systems, no technical specification exists, so the verification of documented properties means checking invariants from inline code comments or examples from user manuals.

An exception to this lack of documentation is software in the telecommunications area, where researchers have a penchant for detailed technical specifications. RIP offers a case study in motivation. Early implementations of distance vector routing were incompatible, so all of the routers running RIP in a domain needed to use the same implementation. Users and implementors were led to correct this problem by providing a specification that would define precise protocols and packet formats. We find below that the resulting standard ([7, 12]) is precise enough to support, without significant supplementation, a very detailed proof of correctness in terms of invariants referenced in the specification. The proved properties are guaranteed to hold of any conformant implementation and of any network of conformant routers. RIP is perhaps better than the average in this respect, since (1) the standard seeks to bind itself closely to its underlying theory, (2) distance vector routing is simpler than some alternative routing approaches, and (3) at this stage, RIP is a highly seasoned standard whose shortcomings have been identified through substantial experience. This is not to say that RIP was already verified by its referenced theory. There are substantial gaps between ([7, 12]) and the asynchronous distributed protocol proved correct in [2]: the algorithm is different in several non-trivial ways, the model is different, and the state maintained is different. Our analysis narrows this gap and extends the results of the theory as applied to the standard version of the protocol.

It is natural to expect that newer protocols, possibly specified in a sequence of draft standards, will have more gaps and will be more likely to evolve. Useful application of formal methods to such projects must 'track' this instability, locating errors or gaps quickly and leveraging other activities like revision of the draft standard and the development of simulations and implementations. To test this agility for our tools and methods we have extended our analysis of RIP to newer applications of distance vector routing in the emerging area of mobile ad hoc networks. Ad hoc networks are networks formed from mobile computers without the use of a centralized authority. A variety of protocols are under development for such networks [21], including many based on distance vector routing ([16, 3, 15, 18]). Requirements for a routing protocol for ad hoc networks are quite different from those of other kinds of networks because of considerations like highly variable connectivity and low bandwidth links. Given the rapid rate of evolution in this area and the sheer number of new ideas, it seems like an appropriate area as a test case for formal methods as part of a protocol design effort.

1.3 Verification Attributes of Routing Protocols

There have been a variety of successful studies of communication protocols. For instance, [13] provides a proof of some key properties of SSL 3.0 handshake protocol [4]. However, most of the studies to date have focused on *endpoint* protocols like SSL using models that involve two or three processes (representing the endpoints and an adversary, for instance). Studies of routing protocols must have a different flavor since a proof that works for two or three routers is not interesting unless it can be generalized. Routing protocols generally have the following attributes which influence the way formal verification techniques can be applied:

1. An (essentially) unbounded number of replicated, simple processes execute concurrently.

2. Dynamic connectivity is assumed and fault tolerance is required.

3. Processes are reactive systems with a discrete interface of modest complexity.

4. Real time is important and many actions are carried out with some timeout limit or in response to a timeout.

Most routing protocols have other attributes such as latencies of information flow (limiting, for example, the feasibility of a global concept of time) and the need to protect network resources. These attributes sometimes make the protocols more complex. For instance, the asynchronous version of the Bellman-Ford protocol is much harder to prove correct than the synchronous version [2], and the RIP standard is still harder to prove correct because of the addition of complicating optimizations intended to reduce latencies.

In this paper we verify protocols using tools that are very general (HOL) or tuned for the verification of communication protocols (SPIN). The tools will be described in Section 2, and the rest of the paper focuses on applications in the case studies. A key technique is *replica generalization*. This technique consists of considering first a router r connected by interfaces to a network N that satisfies a property A. We then prove that r satisfies a property B. In the next step we attempt to prove a property C of r by assuming that properties A and B' hold of N where B' is the property that all of the routers in N satisfy B. A proof is organized as a sequence of replica generalizations. Using this technique and others, we provide a proof of the correctness of RIP in Section 3, proof of a sharp realtime bound on convergence of RIP in Section 4, and proof of path invariants for AODV in Section 5. We offer some conclusions and statistics in the final section.

2 Approaches to Formal Verification

Computer protocols have long been the chosen targets of verification and validation efforts. This is primarily because protocol design often introduces subtle bugs which remain hidden in all but a few *runs* of the protocol. These bugs can nevertheless prove fatal for the system if they occur. In this section, we discuss the complexities involved in verifying network protocols and propose automated tool support for this task. As an example, we consider a simple protocol for leader-election in a network. A variant of this protocol is used for discovering spanning trees in an extended LAN ([19, 20]).

The network consists of n connected nodes. Each node has a unique integer *id*. The node with the least id is called the *leader*. The aim of the protocol is for every node to discover the id of the leader. To accomplish this, each node maintains a *leader-id*: its own estimate of who the leader is, based on the information it has so far. Initially, the node believes itself to be the leader. Every p seconds, each node sends an advertisement containing its leader-id to all its neighbors. On receiving such an advertisement, a node updates its leader-id if it has received a lower id in the message.

The above protocol involves n processes that react to incoming messages. The state of the system consists of the (integer) leader-ids at each process; the only events that can occur are message transmissions initiated by the processes themselves. However, due to the asynchronous nature of the processes, the message transmissions could occur in any order. This means that in any period of p seconds, there could be more than $n!$ possible sequences of events that the system needs to react to. It is easy to see that manual enumeration of the protocol event or state sequences becomes impossible as n is increased. For more complex protocols, manually tracing the path of the protocol for even a single sample trace becomes tedious and error-prone. Automated support for this kind of analysis is clearly required.

A well-known design tool for protocol analysis is simulation. However, to simulate the election protocol, we would first have to fix the network size and topology, then specify the length of the simulation. Finally, we can run the protocol and look at its trace for a given initial state and a single sequence of events. This simulation process, although informative, does not provide a complete verification. A verification should provide guarantees about the behavior of the protocol on all networks, over all lengths of time, under all possible initial states and for every sequence of events that can occur.

We discuss two automated tools that can help provide these guarantees. First, we describe the model-checker SPIN, which can be used to simulate and possibly verify the protocol for a given network (and initial state). We then describe the interactive theorem-prover HOL, which, with more manual effort, can be used to verify general mathematical properties of the protocol in an arbitrary network.

2.1 Model Checking Using SPIN

The SPIN model-checking system ([9, 10]) has been widely used to verify communication protocols. The SPIN system has three main components: (1) the Promela protocol specification language, (2) a protocol simulator that can perform random and guided simulations, and (3) a model-checker that performs an exhaustive state-space search to verify that a property holds under all possible simulations of the system.

To verify the leader-election protocol using SPIN, we first model the protocol in Promela. A Promela model consists of processes that communicate by message-passing along buffered channels. Processes can modify local and global state as a result of an event. The Promela process modeling the leader-election protocol at a single node is as given in Table 1. We then hard-code a

Table 1: Leader Election in Promela

```
#define NODES 3
#define BUF_SIZE 1
chan input[NODES] = [BUF_SIZE] of {int};
chan broadcast = [0] of {int,int};
int leader_id[NODES];

proctype Node (int me; int myid){
    int advert;
    leader_id[me] = myid;
    do
    :: input[me]?advert ->
               if
               :: advert < leader_id[me] ->
                      leader_id[me] = advert
               :: else -> skip
               fi
    :: true -> broadcast!me,leader_id[me]
    od
}
```

network into the broadcast mechanism and simulate the protocol using SPIN. SPIN simulates the behavior of the protocol over a random sequence of events. Viewing the values of the leader-ids over the period of the simulation provides valuable debugging information as well as intuitions about possible invariants of the system.

Finally, we use the SPIN verifier to prove that the election protocol succeeds in a 3-node network. This involves specifying the correctness property in Linear Temporal Logic (LTL). In our case, the specification simply insists that the leader-id at each node eventually stabilizes at the correct id. The verifier then carries out an exhaustive search to ensure that the property is true for every possible simulation of the system. If it fails for any allowed event sequence, the verifier indicates the failure along with the counter-example, which can be subsequently re-simulated to discover a possible bug.

2.2 Interactive Theorem Proving Using HOL

The HOL Theorem Proving System ([5, 8]) is a widely used general-purpose verification environment. The main components of the HOL system are (1) a functional programming language used for specifying functions, (2) Higher-Order Logic used to specify properties about functions, and (3) a proof assistant that allows the user to construct proofs of such properties by using inbuilt and user-defined proof techniques. Both the programming model and the proof environment are very general, capable of proving any mathematical theorem. Designing the proof strategy is the user's responsibility.

In order to model the leader-election protocol in HOL, we need to model processes and message-passing in a functional framework. We take our cue from the reactive nature of the protocol. The input to the protocol is a potentially infinite sequence of messages. The processes can then be considered as functions that take a message as input and describe how the system state is modified. The resulting model is essentially the function in Table 2. Note that the generality

Table 2: State Update Function

```
function Update (state,sender,receiver,mesg,node) =
        if node = receiver then
           if mesg < state(receiver)
           then mesg else state(receiver)
        else state(node)
```

of the programming platform allows us to define the protocol for an arbitrary network in a uniform way.

We then specify the property that we desire from the protocol as a theorem that we wish to prove in HOL.

Theorem 1 *Eventually, every node's leader-id is the minimum of all the node ids in the network.* □

In order to prove this property, we specify some lemmas that must be true of the protocol as well, all of which can be easily encoded in Higher-Order Logic.

Lemma 2 *At each node, the leader-id can only decrease over time.* □

Lemma 3 *If the state of the network is unchanged by a message from node n_1 to node n_2 as well as a message from n_2 to n_1, the leader-ids at n_1 and n_2 must be the same.* □

Lemma 4 *Once a node's leader-id becomes correct, it stays correct.* □

Finally, we construct a proof of the desired theorem. The proof assistant organizes the proof and ensures that the proofs are complete and bug-free. We first prove the lemmas by case analysis on the states and the possible messages at each point in time. Then, Lemmas 2 and 3 are used to prove that the state of the network must 'progress' until all the nodes have the same leader-id. Moreover, since the leader node's leader-id never changes (Lemma 4), all nodes must end up with the correct leader-id. These proofs are carried out in a simple deductive style managed by the proof assistant.

The above proof is just one of many different proofs that could be developed in the HOL system. For example, if instead of correctness, we were interested in proving how long the protocol takes to elect a leader, we could prove the following lemma. Recall that p is the interval for advertisements.

Lemma 5 *If all nodes within a distance k of the leader become correct after t seconds, then all nodes within a distance $(k+1)$ must become correct within $t+p$ seconds.* $\square$

In conjunction with Lemma 4, this provides a more informative inductive proof of the Theorem.

2.3 Model Checking Vs Interactive Theorem Proving

We have described how two systems can address a common protocol verification problem. The two systems clearly have different pay-offs. SPIN offers comprehensive infrastructure for easily modeling and simulating communication protocols and has fixed verification strategies for that domain. On the other hand, HOL offers a more powerful mathematical infrastructure, allowing the user to develop more general proofs. SPIN verifications are generally bound by *memory* and *expressiveness.* HOL verifications are bound by *man-months.*

Our technique is to code the protocol first in SPIN and use HOL to address limits in the expressiveness of SPIN. This is achieved by using HOL to prove *abstractions*, showing properties like: if property P holds for two routers, then it will hold for arbitrarily many routers. Or: advertisements of distances can be assumed to be equal to k or $k+1$. Also, abstraction proofs in HOL were used to reduce the memory demands of SPIN proofs and assure that the SPIN implementation properly reflected the standard. We give examples of these tradeoffs in the case studies and summarize with some statistical data in the conclusions.

3 Stability of RIP

We will assume that the reader is already familiar with the RIP protocol. Its specification is given in ([7, 12]) and a good exposition can be found in [11].

3.1 Formal Terminology

We model the universe $\mathcal{U}$ as a bipartite connected graph whose nodes are partitioned into *networks* and *routers*, such that each router is connected to at least two networks. The goal of the protocol is to compute a table at each router providing, for each network n, the length of the shortest path to n and the next hop along one such path. The protocol is viewed as inappropriate for networks that have more than 15 hops between a router and a destination network, so the hop count is limited to a maximum of 16. A network that is marked as 16 hops away is considered to be unreachable.

Our proof shows that, for each destination d, the routers will all eventually obtain a correct shortest path to d. An entry for d at a router r consists of three parameters:

- $\mathsf{hops}(r)$: current estimate of the distance metric to d (an integer between 1 and 16 inclusively).
- $\mathsf{nextN}(r)$: the next network on the route to d.
- $\mathsf{nextR}(r)$: the next router on the route to d.

Both r and $\mathsf{nextR}(r)$ must be connected to $\mathsf{nextN}(r)$. We say that r *points to* $\mathsf{nextR}(r)$. Initially, routers connected to d must have their metric set to 1, while others must have it set to values strictly greater than 1. Two routers are *neighbors* if they are connected to the same network. The universe changes its state (*i.e.* routing tables) as a reaction to *update messages* being sent between neighboring routers. Each update message can be represented as a triple $(\mathsf{snd}, \mathsf{net}, \mathsf{rcv})$, meaning that the router src sends its current distance estimate through the network net to the router rcv. In some cases this will cause the receiving router to update its own routing entry. An infinite sequence of such messages $(\mathsf{snd}_i, \mathsf{net}_i, \mathsf{rcv}_i)_{i \geq 0}$ is said to be *fair* if every pair of neighboring routers s and r exchanges messages infinitely often:

$$\forall i.\ \exists j > i.\ (\mathsf{snd}_i = s) \ \text{ and } \ (\mathsf{rcv}_i = r).$$

This property simply assures that each router will communicate its routing information to all of its neighbors. Distance to d is defined as

$$D(r) = \begin{cases} 1, & \text{if } r \text{ is connected to } d \\ 1 + \min\{D(s) \mid s \text{ neighbor of r}\}, & \text{otherwise.} \end{cases}$$

For $k \geq 1$, the k-*circle* around d is the set of routers

$$C_k = \{r \mid D(r) \leq k\}.$$

For $1 \leq k \leq 15$, we say that the universe is k-*stable* if the following properties S1 and S2 both hold:

(S1) Every router $r \in C_k$ has its metric set to the actual distance: that is, $\mathsf{hops}(r) = D(r)$. Moreover, if r is not connected to d, it has its next network and next router set the first network and router on some shortest path to d: that is, $D(\mathsf{nextR}(r)) = D(r) - 1$.

(S2) For every router $r \notin C_k$, $\mathsf{hops}(r) > k$.

Given a k-stable universe, we say that a router r at distance $k+1$ from d is $(k+1)$-*stable* if it has an optimal route: that is, $\mathsf{hops}(r) = k+1$ and $\mathsf{nextR}(r) \in C_k$.

3.2 Proof results

Our first goal is to show that RIP indeed eventually discovers all the shortest paths of length less than 16:

Theorem 6 (Correctness of RIP) *For any $k < 16$, starting from an arbitrary state of the universe $\mathcal{U}$, for any fair sequence of update messages, there is a time t_k such that $\mathcal{U}$ is k-stable at all times $t \geq t_k$.* □

In particular, 15-stability will be achieved, which is our original goal. Notice that the result applies to an *arbitrary* initial state. This is critical for the fault tolerance aspect of RIP, since it assures convergence even in the presence of topology changes. As long as the changes are not too frequent, we can apply the theorem to the periods in between them.

Our proof, which we call *the radius proof,* differs from the one described in [2] for the asynchronous Bellman-Ford algorithm. Rather than inducting on estimates for upper and lower bounds for distances, we induct on the the radius of the k-stable region around d. The proof has two attributes of interest:

1. *It states a property about the RIP protocol, rather the asynchrous distributed Bellman-Ford algorithm.* Closer analysis reveals subtle, but substantial differences between the two. In the case of Bellman-Ford, routers keep all of their neighbors' most recently advertised metric estimates, whereas RIP keeps only the best value. Furthermore, the Bellman-Ford metric ranges over the set of all positive integers, while the RIP metric saturates at 16, which is regarded as infinity. Finally, RIP includes certain engineering optimizations, such as split horizon with poison reverse, that do not exist in the Bellman-Ford algorithm.

2. *The radius proof is more informative.* It shows that correctness is achieved quickly close to the destination, and more slowly further away. We exploit this in the next section to show a realtime bound on convergence.

Theorem 6 is proved by induction on k. There are four parts to it:

Lemma 7 *The universe $\mathcal{U}$ is initially 1-stable.* □

Lemma 8 (Preservation of stability) *For any $k < 16$, if the universe is k-stable at some time t, then it is k-stable at any time $t' \geq t$.* □

Lemma 9 *For any $k < 15$ and router r such that $D(r) = k + 1$, if the universe is k-stable at some time t_k, then there is a time $t_{r,k} \geq t_k$ such that r is $(k+1)$-stable at all times $t \geq t_{r,k}$.* □

Lemma 10 (Progress) *For any $k < 15$, if the universe $\mathcal{U}$ is k-stable at some time t_k, then there is a time $t_{k+1} \geq t_k$ such that $\mathcal{U}$ is $(k+1)$-stable at all times $t \geq t_{k+1}$.* □

Lemma 7 is easily proved by HOL and serves as the basis of the overall induction. Lemma 8 is the fundamental safety property, which is also proved in HOL. Parts that can be proved in either tool are typically done in SPIN, since it provides more automation. Lemma 9, the main progress property in the proof, is proved with SPIN, but SPIN can be used only for verification of models for which there is a constant upper bound on the number of states whereas the model from Lemma 9 can, in principle, have an arbitrarily large state space. This problem is solved by finding a finitary property-preserving abstraction of the system and checking the desired property of the abstracted system using SPIN. Proof that the abstraction is indeed property-preserving is done in HOL. The proof as a whole illustrates well how the two systems can prove properties as a team. Interestingly enough, this argument uses the previously-proved lemma about preservation of stability. This is an instance of replica generalization, where proving one invariant allowed us to further simplify (*i.e.* abstract) the system; this, in turn, facilitated the derivation of yet another property. Lemma 10 is the inductive step, which is derived in HOL as an easy generalization of Lemma 9, considering the fact that the number of routers is finite.

4 Sharp Timing Bounds for RIP Stability

In the previous section we proved convergence for RIP conditioned on the fact that the topology stays unchanged for some period of time. We now calculate how big that period of time must be. To do this, we need to have some knowledge about the times at which protocol events must occur. In the case of RIP, we use the following:

Fundamental Timing Assumption There is a value Δ, such that during every topology-stable time interval of the length Δ, each router gets at least one update message from each of its neighbors. $\square$

This is the only assumption we make about timing of update messages. RIP routers normally try to exchange messages every 30 seconds; a failure to receive an update within 180 seconds is treated as a link failure. Thus $\Delta = 3$ minutes satisfies the Fundamental Timing Assumption for RIP.

As in the previous section, we will concentrate on a particular destination network d. Our timing analysis is based on the notion of weak k-stability. For $2 \leq k \leq 15$, we say that the universe $\mathcal{U}$ is *weakly k-stable* if the following conditions hold:

(WS1) $\mathcal{U}$ is $(k-1)$-stable.

(WS2) $\forall r.\ D(r) = k \ \Rightarrow\ (r \text{ is } k{-}\text{stable or } \mathsf{hops}(r) > k)$.

(WS3) $\forall r.\ D(r) > k \ \Rightarrow\ \mathsf{hops}(r) > k$.

Weak k-stability is stronger than $(k-1)$-stability, but weaker than k-stability. The disjunction in (WS2) (which distinguishes weak stability from the ordinary stability) will typically introduce additional complexity in case analyses arising from reasoning about weak stability.

As with k-stability, we have the following:

Lemma 11 (Preservation of weak stability) *For any $2 \leq k \leq 15$, if the universe is weakly k-stable at some time t, then it is weakly k-stable at any time $t' \geq t$.* □

We must also show that the initial state inevitably becomes weakly 2-stable after messages have been exchanged between every pair of neighbors:

Lemma 12 (Initial progress) *If the topology does not change, the universe becomes weakly 2-stable after Δ time.* □

The main progress property says that it takes 1 update interval to get from a weakly k-stable state to a weakly $(k+1)$-stable state. This property is shown in two steps: first we show that condition (WS1) for weak $(k+1)$-stability holds after Δ:

Lemma 13 *For any $2 \leq k \leq 15$, if the universe is weakly k-stable at some time t, then it is k-stable at time $t + \Delta$.* □

and then we show the same for conditions (WS2) and (WS3). The following puts both steps together:

Lemma 14 (Progress) *For any $2 \leq k < 15$, if the universe is weakly k-stable at some time t, then it is weakly $k+1$-stable at time $t + \Delta$.* □

The *radius* of the universe (with respect to d) is the maximum distance from d:

$$R = max\{D(r) \mid r \text{ is a router}\}.$$

The main theorem describes convergence time for a destination in terms of its radius:

Theorem 15 (RIP convergence time) *A universe of radius R becomes 15-stable within $max\{15, R\} \cdot \Delta$ time, assuming that there were no topology changes during that time interval.* □

The theorem is an easy corollary of the preceding lemmas. Consider a universe of radius $R \leq 15$. To show that it converges in $R \cdot \Delta$ time, observe what happens during each Δ-interval of time:

after Δ	weakly 2-stable	(by Lemma 12)
after $2 \cdot \Delta$	weakly 3-stable	(by Lemma 14)
after $3 \cdot \Delta$	weakly 4-stable	(by Lemma 14)
...	...	...
after $(R-1) \cdot \Delta$	weakly R-stable	(by Lemma 14)
after $R \cdot \Delta$	R-stable	(by Lemma 13)

R-stability means that all the routers that are not more than R hops away from d will have shortest routes to d. Since the radius of the universe is R, this includes *all* routers.

An interesting observation is that progress from (ordinary) k-stability to (ordinary) $(k+1)$-stability is not guaranteed to happen in less than $2 \cdot \Delta$ time (we leave this to the reader). Consequently, had we chosen to calculate convergence time using stability, rather than weak stability, we would get a worse upper bound of $2 \cdot (R-1) \cdot \Delta$. In fact, our upper bound is sharp: in a linear topology, update messages can be interleaved in such a way that convergence time becomes as bad as $R \cdot \Delta$. Figure 1 shows an example that consists of k routers and has

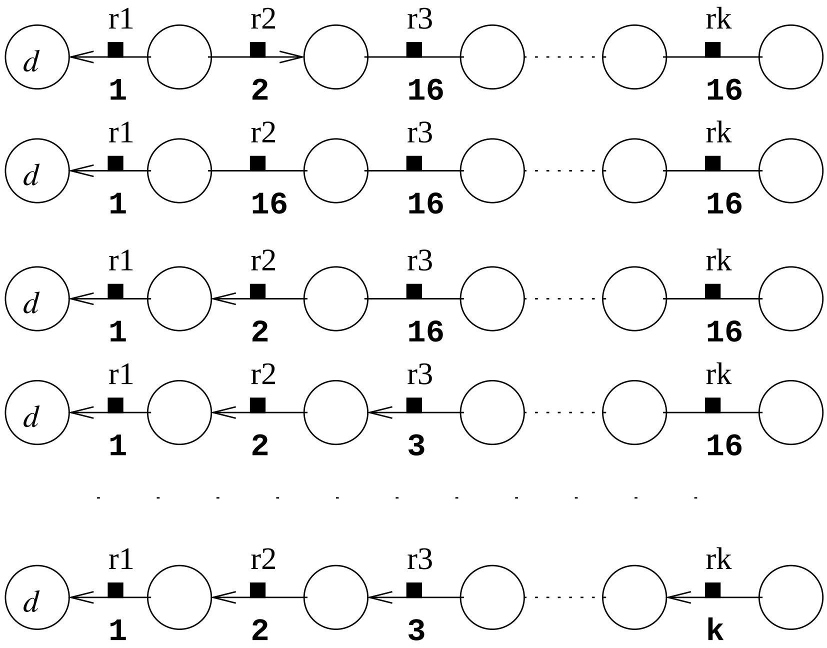

Figure 1: Maximum Convergence Time

the radius k with respect to d. Router r_1 is connected to d and has the correct metric. Router r_2 also has the correct metric, but points in the wrong direction. Other routers have no route to d. In this state, r_2 will ignore a message from r_1, because that route is no better than what r_2 (thinks it) already has. However, after receiving a message from r_3, to which it points, r_2 will update its metric to 16 and lose the route. Suppose that, from this point on, messages are interleaved in such a way that during every update interval, all routers first send their update messages and then receive update messages from their neighbors. This will cause exactly one new router to discover the shortest route during every update interval. Router r_2 will have the route after the second interval, r_3 after the third, ..., and r_k after the k-th. This shows that our upper bound of $k \cdot \Delta$

is reachable.

4.1 Proof methodology

Lemmas 11, 12, and 14 are proved in SPIN (Lemma 13 is a consequence of Lemma 14). Theorem 15 is then derived as a corollary in HOL. SPIN turned out to be extremely helpful for proving properties such as Lemma 14, which involve tedious case analysis. To illustrate this, assuming weak k-stability at time t, let us look at what it takes to show that condition (WS2) for weak $(k+1)$-stability holds after Δ time. ((WS1) will hold because of Lemma 13, but further effort is required for (WS3).)

To prove (WS2), let r be a router with $D(r) = k + 1$. Because of weak k-stability at the time t, there are two possibilities for r: (1) r has a k-stable neighbor, or (2) all of the neighbors of r have $\mathsf{hops} > k$. To show that r will eventually progress into either a $(k+1)$-stable state or a state with $\mathsf{hops} > k+1$, we need to further break the case (2) into three subcases with respect to the properties of the router that r points to: (2a) r points to $s \in C_k$ (the k-circle), which is the only neighbor of r from C_k, or (2b) r points to $s \in C_k$, but r has another neighbor $t \in C_k$ such that $t \neq s$, or (2c) r points to $s \notin C_k$. Each of these cases, branches into several further subcases based on the relative ordering in which r, s and possibly t send and receive update messages.

Doing such proofs by hand is difficult and prone to errors. Essentially, the proof is a deeply-nested case analysis in which *final* cases are straight-forward to prove—an ideal job for a computer to do! Our SPIN verification is divided into four parts accounting for differences in possible topologies. Each part has a distinguished process representing r and another processes modeling the environment for r. An environment is an abstraction of the 'rest of the universe'. It generates all message sequences that could possibly be observed by r. Sometimes a model can be simplified by allowing the environments to generate some sequences that are not possible in reality. This does not affect the confidence in *positive* verification answers, since any invariant that holds in a less constrained environment also holds in a more constrained one. SPIN considered more cases than a manual proof would have required, 21,487 of them altogether for Lemma 14, but it checked these in only 1.7 seconds of CPU time. Even counting set-up time for this verification, this was a significant time-saver. The resulting proof is probably also more reliable than a manual one. We summarize similar analyses for our other results in the conclusions.

5 Verifying Properties of AODV

The mobile, ad hoc network model consists of a number of mobile *nodes* capable of communicating with their *neighbors* via broadcast or point-to-point *links*. The mobility of the nodes and the nature of the broadcast media make these links temporary, low bandwidth, and highly lossy. Nodes can be used as routers to forward packets. At any point in time, the nodes can be thought of as forming a connected ad hoc network with multi-hop paths between nodes. In this environment, the goal of a routing protocol is to discover and maintain routes between nodes. The essential requirements of such a protocol would be to (1) react quickly to topology changes and discover new routes, (2) send a minimal amount of control information, and (3) maintain 'good' routes so that data packet throughput is maximized.

Distance-vector protocols like RIP are probably satisfactory for requirements (2) and (3). However, the behavior of these protocols is drastically altered in the presence of topology changes. For example, in RIP, if a link goes down, the network could take *infinite* time (15 update periods) to adapt to the new topology. This is because of the presence of routing loops in the network, resulting in the so called *counting-to-infinity* problem. If routing loops could be avoided, distance-vector protocols would be strong candidates for routing in mobile, ad-hoc networks. There have been a variety of proposals for distance vector protocols that avoid loop formation. We consider AODV [18], a recently-introduced protocol that aims to minimize the flow of control information by establishing routes only *on demand.* Our analysis is based on the version 2 draft specification [17].

In a network running AODV, every node n maintains two counters: a *sequence number* ($\mathsf{seqno}(n)$) and a *broadcast id* ($\mathsf{broadcast\_id}(n)$. The sequence number is incremented when the node discovers a local topology change, while the broadcast id is incremented every time the node makes a route-discovery broadcast. The node also maintains a *routing table* containing routes for each destination d that it needs to send packets to. The route contains the *hop-count* ($\mathsf{hops}_d(n)$) to d and the *next* node ($\mathsf{next}_d(n)$) on the route. It also contains the last known sequence number ($\mathsf{seqno}_d(n)$) of the destination, which is a measure of the freshness of the route.

When a source node s needs to discover a route to a destination d, it broadcasts a *route-request* (RREQ) packet to all its neighbors. The RREQ packet contains the following fields:

$$\langle s, \mathsf{seqno}(s), \mathsf{hop\_cnt}, d, \mathsf{seqno}_d(s), \mathsf{broadcast\_id}(s)\rangle.$$

The RREQ is thought of as requesting a route from s to d at least as fresh as the last known destination sequence number ($\mathsf{seqno}_d(s)$). At the same time, the packet informs the recipient node that it is $\mathsf{hop\_cnt}$ hops away from the node s, whose current sequence number is $\mathsf{seqno}(s)$. Consequently, a node that receives a new RREQ re-broadcasts it with an incremented $\mathsf{hop\_cnt}$ if it cannot satisfy the request. Using the source information in the packet, every such intermediate node also sets up a reverse-route to s through the node that sent it the RREQ. The $\mathsf{broadcast\_id}$ is used to identify (and discard) multiple copies of an RREQ received at a node.

When the RREQ reaches a node that has a fresh enough route to d, the node unicasts a *route-reply* (RREP) packet back to s. Remember that the nodes that forwarded the RREQ have set up a reverse route that can be used for this unicast. The RREP packet contains the following fields:

$$\langle s, d, \mathsf{seqno}(d), \mathsf{hop\_cnt}, \mathsf{lifetime}\rangle.$$

As in the RREQ case, this message provides information ($\mathsf{hop\_cnt}, \mathsf{seqno}(d)$) that a recipient node can use to set up a route to d. The route is assumed to be valid for $\mathsf{lifetime}$ milliseconds. Any node receiving the RREP updates its route to d and passes it along towards the requesting source node s after incrementing the $\mathsf{hop\_cnt}$. In case a receiving node already has a route to d, it uses the route with the maximum $\mathsf{seqno}(d)$ and then the one with the least $\mathsf{hop\_cnt}$.

The above process can be used to establish routes when the network is stable. However, when a link along a route goes down, the source node needs to recognize this change and look for an alternative route. To achieve this, the node immediately upstream of the link sends RREP messages to all its neighbors that have been actively using the route. These RREP messages have the $\mathsf{hop\_cnt}$ set to INFINITY (255) and $\mathsf{seqno}(d)$ set to one plus the previously known sequence number. This forces the neighbors to recognize that the route is unavailable, and they then forward the message to their active neighbors until all relevant nodes know about the route failure. Note however, that this process depends on the nodes directly connected to a link being able to detect its unavailability. Link failure detection is assumed to be handled by some link-layer mechanism. If not, the nodes can run the *hello* protocol, which requires neighbors to send each other periodic 'hello' messages indicating their availability.

5.1 Path Invariants

As mentioned before, an essential property of AODV for handling topological changes is that it is loop-free. A hand proof by contradiction is given in [18]. We provide an automated proof of this fact as a corollary of the preservation of the key path invariant of the protocol. The invariant is also used to prove route validity.

First, we model AODV in SPIN by Promela processes for each node. As described earlier, the process needs to maintain state in the form of a broadcast-id, a sequence number and a routing table. In the following, we write $\mathsf{seqno}_d(d)$ to denote d's sequence number ($\mathsf{seqno}(d)$). The process needs to react to several events, possibly updating this state. The main events are neighbour discovery, data or control (RREP/RREQ) packet arrival and timeout events like link failure detection and route expiration. It is relatively straight-forward to generate the Promela processes from [17].

Then, we prove that the following is an invariant (over time) of the AODV process at a node n, for every destination d.

Theorem 16 *If* $\mathsf{next}_d(n) = n'$, *then*

1. $\mathsf{seqno}_d(n) \leq \mathsf{seqno}_d(n')$, *and*
2. $\mathsf{seqno}_d(n) = \mathsf{seqno}_d(n') \Rightarrow \mathsf{hops}_d(n) > \mathsf{hops}_d(n')$. □

This invariant has two important consequences:

1. (Loop-Freedom) Consider the network at any instant and look at all the routing-table entries for a destination d. Any data packet traveling towards d would have to move along the path defined by the next_d pointers. However, we know from Theorem 16 that at each hop along this path, either the sequence number must increase or the hop-count must decrease. In particular, a node cannot occur at two points on the path. This guarantees loop-freedom for AODV.

2. (Route Validity) Loop-freedom in a finite network guarantees that data paths are finite. This does not guarantee that the path ends at d. However, if all the sequence numbers along a path are the same, hop-counts must strictly decrease (by Theorem 16). In particular, the last node n_l on the path cannot have hop-count INFINITY (no route). But since n_l does not have a route to d, it must be equal to d.

To prove Theorem 16, we first prove the following properties about the routing table at each node n, now considered as a function of time.

Lemma 17 *If* $t_1 \leq t_2$*, then*
$\mathsf{seqno}_d(n)(t_1) \leq \mathsf{seqno}_d(n)(t_2)$. □

Lemma 18 *If* $t_1 \leq t_2$ *and*
$\mathsf{seqno}_d(n)(t_1) = \mathsf{seqno}_d(n)(t_2)$*, then*
$\mathsf{hops}_d(n)(t_1) \geq \mathsf{hops}_d(n)(t_2)$. □

Suppose $\mathsf{next}_d(n)(t) = n'$. Then we define lut (last update time), to be the last time before t, when $\mathsf{next}_d(n)$ *changed* to n'. We claim that following lemma holds for times t and lut:

Lemma 19 *If* $\mathsf{next}_d(n)(t) = n'$*, then*

1. $\mathsf{seqno}_d(n)(t) = \mathsf{seqno}_d(n')(\mathsf{lut})$*, and*

2. $\mathsf{hops}_d(n)(t) = 1 + \mathsf{hops}_d(n')(\mathsf{lut})$. □

The lemmas are proved in SPIN using the Promela model described earlier. Lemmas 17 and 18 can be proved without restrictions on the events produced by the environment. Lemma 19 is trickier and requires the model to record the incoming $\mathsf{seqno}_d(n')$ and $\mathsf{hops}_d(n')$ whenever the protocol decides to change $\mathsf{next}_d(n)$ to n'. This is easily done by the addition of two variables. Subsequently, Lemma 19 is also verified by SPIN.

The proof that the three lemmas together imply Theorem 16 involves standard deductive reasoning which we did in HOL.

5.2 Failure Conditions

In the previous section, we described a Promela model for AODV and claimed that it is straight-forward to develop the protocol model from the standard specification in [17]. While it is clear what the possible events and corresponding actions are, the Internet Draft does leave some boundary conditions unspecified. In particular, it does not say what the initial state at a node must be.

We assume a 'reasonable' initial state, namely one in which the node's sequence number and broadcast-id are 0 and the routing table is empty. The proofs in the previous section were carried out with this initial-state assumption. However, it is unclear if this assumption is necessary. Importantly, would an AODV implementation that chose to have some *default* route and sequence-number fail to satisfy some crucial properties?

We approach this problem by identifying the *key invariants* that must be satisfied by any strategy that the node may use on reboot. We choose the invariant in Theorem 16 as a target of our efforts, since violating this invariant *may* result in breaking the loop-freedom property. A newly-initialized node n_i can break the invariant in exactly two ways:

1. $n_i = n$: This means that the node has initialized with a route for d that goes through n'
2. $n_i = n'$: Some node (n) has a route through n_i even though n_i has just come up. This implies that n_i had been previously active before it failed and the node failure was not noticed by n.

The key choices left to the implementor are:

1. whether node failures are detected by all neighbors,
2. whether nodes can initialize with a (default) route, and
3. whether nodes can initialize with an arbitrary sequence number.

However, any choice that the programmer makes must comply with the invariant proved in Theorem 16. Keeping this in mind, we analyze all the possible choices. First, we find that failure-detection (1) is necessary. Otherwise, it is possible that $\mathsf{next}_d(n) = n_i$ and n_i has no route to d. So, Part (1) of the invariant in Theorem 16 is immediately broken and loops may be formed. For instance, n_i may soon look for a route to d and accept the route that n has to offer, thus forming a loop.

Assuming failure detection, the safest answer to (2) is to disallow any initial routes. This ensures that the invariant cannot be violated and therefore loop-freedom will be preserved. On the other hand, if initial routes are allowed, multiple failures would make the invariant impossible to guarantee.

Choice (3) comes into play only at the destination d. If d itself is initialized, we assume that it does not have a route for itself. But then both choices for (3) obey the invariant in the presence of failure detection (1). Moreover, d can never be a member of the routing loop. This means that in the choice of (2) above,

Table 3: Protocol Verification Effort

Task	HOL	SPIN
Modeling RIP	495 lines, 19 defs, 20 lemmas	141 lines
Proving Lemma 8 Once	9 lemmas, 119 cases, 903 steps	
Proving Lemma 8 Again	29 lemmas, 102 cases, 565 steps	207 lines, 439 states
Proving Lemma 9	Reuse Lemma 8 Abstractions	285 lines, 7116 states
Proving Lemma 11	Reuse Lemma 8 Abstractions	216 lines, 1019 states
Proving Lemma 12	Reuse Lemma 8 Abstractions	221 lines, 1139 states
Proving Lemma 14	Reuse Lemma 8 Abstractions	342 lines, 21804 states
Modeling AODV	95 lines, 6 defs	302 lines
Proving Lemma 17		173 lines, 5106 states
Proving Lemma 18		173 lines, 5106 states
Proving Lemma 19		157 lines, 721668 states
Proving Theorem 16	4 lemmas, 2 cases, 5 steps	

we would be safe if $\mathsf{next}_d(n_i) = d$, irrespective of what the invariant suggests. This analysis leads to the following:

Theorem 20 *For AODV to be loop-free, it is sufficient that*

1. *node failures are detected by all neighbors, and*
2. *in the initial state for a node n_i:*
   (a) *there is no route for n_i itself, and*
   (b) *for any other destination d, either n_i has no route for d, or $\mathsf{next}_d(n_i) = d$.* □

Any implementation of AODV should conform to the above Theorem. For example, a simple way to guarantee node-failure detection is to insist that nodes remain silent for a fixed time after coming up, until they can be sure that their previous absence has been detected. Ensuring the rest of the conditions is straight-forward.

6 Conclusion

This paper provides the most extensive automated mathematical analysis of a class of routing protocols to date. Our results show that it is possible to provide formal analysis of correctness for routing protocols from IETF standards and drafts with reasonable effort and speed, thus demonstrating that these techniques can effectively supplement other means of improving assurance such as manual proof, simulation, and testing. Specific technical contributions include: the first proof of the correctness of the RIP standard, statement and automated proof of a sharp realtime bound on the convergence of RIP, and an automated proof of loop-freedom for AODV.

Table 3 summarizes some of our experience with the complexity of the proofs in terms of our automated support tools. The complexity of an HOL verification for the human verifier is described with the following statistics measuring things written by a human: the number of *lines* of HOL code, the number of *lemmas* and *definitions*, and the number of proof *steps*. Proof steps were measured as the number of instances of the HOL construct `THEN`. The HOL automated contribution is measured by the number of *cases* discovered and managed by HOL. This is measured by the number of `THENL`'s, weighted by the number of elements in their argument lists. The complexity of SPIN verification for the human verifier is measured by the number of *lines* of Promela code written. The SPIN automated contribution is measured by the number of *states* examined and the amount of *memory* used in the verification. As we mentioned before, SPIN is memory bound; each of the verifications took less than a minute and the time is generally proportional to the memory. Most of the lemmas consumed the SPIN-minimum of 2.54MB of memory; Lemma 19 required 22.8MB. The figures were collected for runs on a lightly-loaded Sun Ultra Enterprise with 1016MB of memory and 4 CPU's running SunOS 5.5.1. The tool versions used were HOL90.10 and SPIN-3.24. We carried out parallel proofs of Lemma 8, the Stability Preservation Lemma, using HOL only and HOL together with SPIN.

Extensions of the results in this paper are possible in several areas: additional protocols, better tool support, and techniques for proving other kinds of properties. We could prove correctness properties similar to the ones in this paper for other routing protocols, including other approaches like link state routing [1]. It would be challenging to prove OSPF [14] because of the complexity of the protocol specification, but the techniques used here would apply for much of what needs to be done. We do intend to pursue better tool support. In particular, we are interested in integration with simulation and implementation code. This may allow us to leverage several related activities and also improve the conformance between the key artifacts: the standard, the SPIN program, the HOL invariants and environment model, the simulation code, and, of course, the implementation itself. One program analysis technology of particular interest is *slicing,* since it is important to know which parts of a program might affect the values in messages. We are also interested in how to prove additional kinds of properties such as security and quality of service (including reservation assurances). Security is particularly challenging because of the difficulty in modeling secrecy precisely.

Acknowledgments

We would like to thank the following people for their assistance and encouragement: Roch Guerin, Elsa L. Gunter, Luke Hornof, Sampath Kannan, Insup Lee, Charles Perkins, and Jonathan Smith. This research was supported by NSF Contract CCR-9505469, and DARPA Contract F30602-98-2-0198.

References

[1] Bhargav Bellur and Richard G. Ogier. A reliable, efficient topology broadcast protocol for dynamic networks. In *IEEE Infocomm*. IEEE, 1999.

[2] Dimitri P. Bertsekas and Robert Gallager. *Data Networks*. Prentice Hall, 1991.

[3] C.-C. Chiang. Routing in clustered multihop, mobile wireless networks with fading channel. In *Proceedings of IEEE SICON '97*, pages 197–211, April 1997.

[4] Alan O. Freier, Philip Karlton, and Paul C. Kocher. *Secure Socket Layer*. IETF Draft, November 1996. `home.netscape.com/ eng/ ssl3`.

[5] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[6] Constance Heitmeyer, James Kirby, and Bruce Labaw. Applying the SCR requirements method to a weapons control panel: An experience report. In *Formal Methods in Software Practice*. ACM SIGSOFT, March 1998.

[7] C. Hendrick. Routing information protocol. RFC 1058, IETF, June 1988.

[8] Home page for the HOL interactive theorem proving system. `www.cl.cam.ac.uk/ Research/ HVG/ HOL`.

[9] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[10] Gerard J. Holzmann. The SPIN model checker. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[11] Christian Huitema. *Routing in the Internet*. Prentice Hall, 1995.

[12] G. Malkin. *RIP Version 2 Carrying Additional Information*. IETF RFC 1388, January 1993.

[13] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *Seventh USENIX Security Symposium*, pages 201–216. USENIX, San Antonio, 1998.

[14] J. Moy. OSPF version 2. RFC 1583, IETF, March 1994.

[15] Shree Murthy and J.J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *ACM Mobile Netowrks and Applications Journal*, October 1996. Special Issue on Routing in Mobile Communication Networks.

[16] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. *Computer Communications Review*, pages 234–244, October 1994.

[17] Charles E. Perkins and Elizabeth M. Royer. Ad hoc on demand distance vector (AODV) routing. Internet-Draft Version 2, IETF, March 1998.

[18] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computer Systems and Applications*, pages 90–100, February 1999.

[19] R. Perlman. An algorithm for distributed computation of spanning trees in an extended LAN. In *Proceedings of the Ninth Data Communications Symposium*, pages 44–53, September 1985.

[20] R. Perlman. *Interconnections: Bridges and Routers*. Addison-Wesley, 1992.

[21] Elizabeth M. Royer and Chai-Keong Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*, pages 46–55, April 1999.