Routing Information Protocol in HOL/SPIN

Karthikeyan Bhargavan, Carl A. Gunter, and Davor Obradovic*

University of Pennsylvania

Abstract. We provide a proof using HOL and SPIN of convergence for the Routing Information Protocol (RIP), an internet protocol based on distance vector routing. We also calculate a sharp realtime bound for this convergence. This extends existing results to deal with the RIP standard itself, which has complexities not accounted for in theorems about abstract versions of the protocol. Our work also provides a case study in the combined use of a higher-order theorem prover and a model checker. The former is used to express abstraction properties and inductions, and structure the high-level proof, while the latter deals efficiently with case analysis of finitary properties.

1 Introduction

The high connectivity on which the Internet relies is enabled by scalable and robust protocols that enable routers connecting different physical networks to forward packets toward destinations described in a uniform addressing system. The first Internet routing protocols were based on distance vector routing, which uses information about distance and direction to a destination to route packets. The first such protocol standardized by the Internet Engineering Task Force (IETF) was the Routing Information Protocol (RIP), and this protocol remains in widespread use today. Although the correctness of distance vector routing has been proved for theoretical versions of the algorithm, the RIP standard itself has never been proved to have some of the properties it is expected to possess. Since there exist non-trivial differences between the abstract version and the standard itself, proofs of some key properties of the standard are worthwhile. In this paper we carry out the proof of convergence using a combination of the HOL [5,9] higher-order theorem prover and the SPIN model checker [10,17]. The automated assistance reduces the burden of case analysis in parts of the standard where manual analysis would prove tedious. Moreover, the HOL/SPIN proof provides high confidence for RIP and insights into the techniques needed to address other routing protocols, most of which are more complex than RIP.

Routing protocols are meant to be robust with respect to failures of links and routers. If there is a failure then the routers communicate this information and routing tables are updated to route around the failed link or router. This process takes some time since routers cannot possess instantaneous global knowledge

* Email addresses: bkarthik@saul.cis.upenn.edu, gunter@cis.upenn.edu, davor@saul.cis.upenn.edu

of network characteristics. They therefore pass information that is incomplete and, if the protocol has the right characteristics, they eventually *converge* on a suitable set of alternative routes. We have two results: we show that the RIP protocol will converge after a failure, and we calculate a sharp realtime bound on the time this will take as a function of the radius of the network. Both results are based on assumptions about network reliability and timing assumptions specified in the RIP protocol.

The first proof concerns the convergence of the asynchronous distributed Bellman-Ford protocol as specified in the IETF RIP standard [8, 12]. The classic proof of a 'pure' form of the protocol is given in [1]. 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 makes the theory better match the standard, and hence also its implementations. Our proof also uses a different technique from the one in [1], providing some noteworthy properties about network stability.

Our second proof 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. From our first proof of convergence, it is easy to see that this 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 supplemented by a new invariant. Thus, if a network has a maximum radius of 5 around 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.

The basis of our verification is the RIP standard. 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, leading to the first version of the standard [8]. In time this standard was revised to a second version [12]. At the level of abstraction we use here, our proof is applicable to both of these versions.

There have been a variety of successful formal studies of communication protocols. However, most of the studies to date have focused on *endpoint* protocols (that is, protocols between pairs of hosts) using models that involve two or three processes (representing the endpoints, or 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 [1], and the RIP standard is still harder to prove correct because of the addition of complicating optimizations intended to reduce latencies.

Following this introduction we give a description of the Routing Information Protocol as specified in its standard. We then describe our formalization of RIP in HOL and SPIN. In the fourth and fifth sections we show the convergence of RIP and derive a sharp realtime bound for the convergence. In the sixth section we provide some analysis of our methodology including a discussion of the benefits of automation and some crude measurements of the complexity of the proofs as viewed by the automated tools and the person carrying out the verification respectively. Our final section summarizes the conclusions.

2 Routing Information Protocol

The RIP protocol specification is given in [8, 12] and a good exposition can be found in [11]. We start by describing the general networking environment and the task of a routing protocol. Then we give a brief description of the RIP protocol, including its pseudocode (Appendices A.1, A.2). Finally, we discuss differences between the standard and the underlying theory and the way they affect protocol requirements.

2.1 Routing in Internetworks

An *internet* is a family of *networks* connected by *routers*. Figure 1 illustrates an internetwork with four networks (shown as clouds) and four routers (shown as black squares). The goal of the routers is to forward packets between hosts (shown as circles) that are attached to the networks. The routers use *routing tables* which they develop through running a distributed *routing protocol*. Packets from hosts travel in *hops* across networks linked by routers. Each router chooses a link on which to forward the packet based on the packet's destination address and other parameters. In order to be able to make good forwarding decisions, routers need to maintain partial topology information in the routing tables.

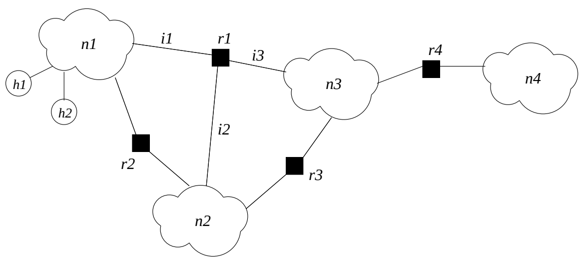

Fig. 1. An Internet

The aim of a routing protocol is to establish a procedure for updating these tables. In most cases, routing information can be exchanged only *locally* (i.e. between neighboring routers). However, the overall goal of a routing protocol is to establish good *global* paths (between distant hosts on the internet). An *interface* is the link between a router and a network. In this example, router $r1$ has interfaces $i1$, $i2$ and $i3$, which connect it to the networks $n1$, $n2$ and $n3$ respectively. Hosts $h1$ and $h2$ belong to the network $n1$. Routers are said to be *neighbors* if they have interfaces to a common network. In our example, all routers are neighbors of $r1$, but $r2$ and $r4$ are not neighbors.

2.2 Routing Information Protocol

Each RIP router maintains a routing table. A routing table contains one entry per destination network, representing the current best route to the destination. An entry corresponding to destination d has the following fields:

- hops: number of hops to d (i.e. total number of routers that a message sent along that route traverses before reaching the network d - this includes the router where this entry resides). This is sometimes called a *metric* for d.
- nextR: next router along the route to d.
- nextIface: the interface that will be used to forward packets addressed to d. It uniquely identifies the next network along the route.

Routers periodically *advertise* their routing tables to their neighbors. Upon receiving an advertisement, the router checks whether any of the advertised routes can be used to improve current routes. Whenever this is the case, the router updates its current route to go through the advertising neighbor. Routes are compared exclusively by their length, measured in the number of hops.

The value of hops must be an integer between 1 and 16, where 16 has the meaning of *infinity* (a destination with hops attribute set to 16 is considered unreachable). Hence, RIP will not be appropriate for internets that contain a router and a destination network that are more than 15 hops apart from each other.

Appendices A.1 and A.2 show pseudocode for RIP. A router advertises its routes by broadcasting RIP packets to all of its neighbors. A RIP packet contains a list of (destination, hops)-pairs. A receiving router compares its current metric for destination to $(1 + \mathsf{hops})$, which is the metric of the alternative route, and updates the corresponding routing entry if the alternative route is shorter. There is one exception to this rule—if the receiving router has the advertising router as nextR for the route, it adopts the alternative route regardless of its metric.

Normally, a RIP packet contains information that reflects the advertising router's own routing table. This rule has an exception too—routers do not advertise routes on the interfaces through which they had been learned. Precisely, if a route is learned over the interface i, it should be advertised on that interface with hops set to 16 (infinity). This rule is called *split horizon with poisoned reverse* and its purpose is to prevent creation of small routing loops.

Each routing table entry has a timer expire associated with it. Every time an entry is updated (or created), expire is re-set to 180 seconds. Routers try to advertise every 30 seconds, but due to network failures and congestion some advertisements may not get through. If a route has not been refreshed for 180 seconds, the router will assume that there was a link failure, the destination will be marked as unreachable and a special garbageCollect timer will be set to 120 seconds. If this timer expires before the entry gets updated, the route is expunged from the table.

2.3 Standard vs. Theory

The mathematical theory behind RIP is described in [1] as the Asynchronous Distributed Bellman-Ford Algorithm (ADBF). In the ADBF model, at every point in time, a router is either idle, sending an advertisement, or receiving an advertisement. The routing table is updated upon receiving an advertisement. Details of the proof that ADBF finds shortest routes are presented in [1].

An interesting question is: 'Can we use (essentially) the same proof to show that RIP protocol converges to the set of shortest routes?' It turns out that the answer is quite certainly 'no'. Although motivated by the ADBF, RIP standard [8, 12] differs from it in several important details:

- ADBF has 'more powerful bookkeeping'. In RIP, routers keep track of only one (current best) route to each destination. On the other hand, ADBF nodes keep, for each destination, the most recent routes *through each of the neighbors*. Correspondingly, this would be reflected in the pseudocode (Appendices A.1, A.2) by all subscript indices becoming *(dest,neighbor)*, instead of just *dest*. This makes ADBF more flexible, which comes at the expense of maintaining a larger data structure.
- RIP has 'blind' updates. As a consequence of the previous difference, RIP routers need to separately handle the case when an advertisement is received from a neighbor which is already nextR for the route. In this case, the receiving router can do nothing better than blindly accept the advertised route, regardless of its length. ADBF does not have this special case.

– RIP's route length is bounded. RIP can handle routes of at most 15 hops. Distances of 16 or more hops are all considered equivalent to infinity. This is a practical optimization intended to balance the tradeoff between quicker loop elimination and greater range for routing information propagation.
– RIP has the *split horizon with poisoned reverse* rule. This is another engineering optimization, not present in ADBF.

The first of the above gaps alone would be enough to make proofs of convergence requirements for RIP substantially different from proofs for ADBF. Besides matching the RIP setting closely, our proof technique also gives useful insights about the speed of propagation of updates, which can be used for establishing timing bounds for convergence.

3 Formal Specification of RIP

In the previous section, we gave a short description of the RIP standard along with its pseudo-code. In this section, we present a formal specification of the protocol that can be analyzed by HOL90 and SPIN. First, we make some simplifications of the protocol:

1. We observe that RIP (Appendices A.1, A.2) operates independently for every destination, with no interaction between the state or events associated with different destinations. This means that we need to specify and prove the protocol only for a single destination and the result will hold for the general version as well.
2. We only analyze the protocol *in between* topology changes. When the protocol starts, it may have *any* sound state to begin with. However, once it has started, one must give the protocol a reasonable period of time to converge. So we assume that there are no topology changes in the lifetime of the analysis. Under this assumption, the protocol indeed converges as we show in Section 4. Moreover, in Section 5, we precisely characterize the time period for which there must be no topology changes to guarantee convergence.
3. We abstract away from actual timing constraints. If topology changes are ruled out, routes cannot be expired ($expire_{dest}$) or deleted ($garbageCollect_{dest}$). So the only timing constraint left is the time interval between periodic broadcasts of advertisements. We model this by (a) enabling a router to broadcast advertisements at any time (a safe abstraction), and (b) adding a fairness assumption to the broadcast sequence.

We next specify RIP in HOL for analysis in the HOL90 theorem prover. Then we model the protocol in Promela, the specification language for the SPIN model-checker. The Promela modeling is straight-forward: it simply involves rewriting the pseudo-code in terms of Promela's C-like syntax, and SPIN's event semantics. The HOL specification is more involved, since we need to transform the pseudo-code into a functional specification.

3.1 RIP in HOL

For a RIP router, the universe $\mathcal{U}$ is a bipartite, connected graph whose nodes are partitioned into *networks* and *routers*, connected through *interfaces*. Routers are always connected to at least two networks. We specify networks and routers using distinct uninterpreted type variables: $'\mathsf{network}$, $'\mathsf{router}$. Now any specific universe can be described simply by a function $\mathsf{conn} : '\mathsf{router} \to '\mathsf{network} \to \mathsf{bool}$, that describes the interfaces—which routers and networks are connected with each other. A function conn describes a *valid* universe $\mathcal{U}$ if (a) conn connects each router to at least 2 networks, and (b) conn describes a connected graph.

When the RIP protocol starts operating in a universe $\mathcal{U}$, it is given as input a valid conn function, describing the topology of $\mathcal{U}$, and an initial state s_0. The protocol then seeks to compute paths from each router to the destination d. We describe the HOL specification in three steps: (1) the state of the protocol, along with an initial state assumption, (2) the processes that change the state and pass messages to each other, and (3) the semantics of these processes in HOL, and typical properties they are expected to satisfy.

Protocol State The goal of RIP is to compute an optimal path at each router r to the destination network d. The path is described by a *routing entry*: the number of hops to d, the next router (nextR) along this path, and the network between r and nextR (nextN). RIP only computes paths of length less than 16; destinations more than 16 hops away are considered unreachable.

The protocol state consists of a table of the current routing entries at each router r, which we call the routing table (rtable). A protocol state is defined as as a 3-tuple $s : \mathsf{rtable}$ whose components are $\mathsf{hops} : '\mathsf{router} \to \mathsf{num}$ and $\mathsf{nextN} : '\mathsf{router} \to '\mathsf{network}$ and $\mathsf{nextR} : '\mathsf{router} \to '\mathsf{router}$. In addition, we want all protocol states to be *sound*, where soundness is defined as follows:

Definition 1 (Soundness). *A protocol state* $s = (\mathsf{hops}, \mathsf{nextN}, \mathsf{nextR})$ *of a universe described by a valid* conn *is said to be sound with respect to* d *if*

1. $\forall r : '\mathsf{router}. \mathsf{conn}\ r\ (\mathsf{nextN}(r)) \wedge \mathsf{conn}\ (\mathsf{nextR}(r))\ (\mathsf{nextN}(r))$
2. $\forall r : '\mathsf{router}. 1 \leq (\mathsf{hops}(r)) \leq 16$
3. $\forall r : '\mathsf{router}. (\mathsf{conn}\ r\ d) \Rightarrow (\mathsf{hops}(r) = 1) \wedge (\mathsf{nextN}(r) = d) \wedge (\mathsf{nextR}(r) = r)$
4. $\forall r : '\mathsf{router}. \neg(\mathsf{conn}\ r\ d) \Rightarrow (\mathsf{hops}(r) > 1) \wedge (\mathsf{nextN}(r) \neq d) \wedge (\mathsf{nextR}(r) \neq r)$

We stipulate that the *initial state* of the protocol, s_0, must be sound. Observe that soundness really has to do with the 'local' connections at a router, which are typically configured by mechanisms external to RIP. By stipulating that the initial state is sound, we require that the router is never deluded about its local topology, otherwise there is no guarantee that it will ever discover global path information. Put another way, if the system ever gets into an unsound state, convergence cannot be guaranteed. Note however, that we can only assume the initial state to be sound, we need to prove that all succeeding states will remain sound under RIP.

Processes We represent different event handlers in the protocol by different processes; they typically perform different kinds of actions and may do so in parallel. As a result, there are three kinds of processes in the universe: each router r has an *advertising process* (generating advertisements), and a *routing process* (handling packet reception), and at each network net there is a *network process* (performing broadcasts).

The advertising process persistently broadcasts route advertisements on each of its connected networks. Each such advertisement is a tuple $(\mathsf{src}, \mathsf{hopcount})$, saying that the broadcasting router src, knows of a path of length $\mathsf{hopcount}$ to the destination d. Suppose the protocol state is $s = (\mathsf{hops}, \mathsf{nextN}, \mathsf{nextR})$, then the $\mathsf{hopcount}$ advertised by src on net may have the following values:

- if $\mathsf{net} = \mathsf{nextN}(\mathsf{src})$, then $\mathsf{hopcount} = 16$ (Infinity);
- otherwise, $\mathsf{hopcount} = \mathsf{hops}(\mathsf{src})$.

When an advertisement is to be broadcast on network net, it is handed over to the network process for net. The network process executes the broadcast by attempting to deliver the incoming advertisement to all routers rcv connected to the net. We do not assume that the network is reliable in any way, so it may not deliver the advertisement to any router, or it may deliver it to some of the routers in an arbitrary order. However, we make the following assumptions

- *Fairness*: the network cannot ignore a router forever. So in any execution of the network net, if a router src sends advertisements infinitely often, and rcv is another router connected to net, the network process must deliver src's advertisements to rcv infinitely often.
- *Zero Delay*: We assume that if the network does deliver an advertisement, it does so instantaneously.

We call the tuple $(\mathsf{src}, \mathsf{net}, \mathsf{rcv}, \mathsf{hopcount})$, corresponding to the delivery of an advertisement, an *advertisement event*. Observe that the unreliability of the networks in conjunction with the persistent broadcasts of the advertisement processes allows many possible sequences of advertisement events. In fact, the network and advertising processes can generate every possible sequence of $(\mathsf{src}, \mathsf{net}, \mathsf{rcv})$ tuples in advertisement events, subject to the fairness assumption and the fact that src and rcv must both be connected to net. The only advertisement field that depends on the network state is the $\mathsf{hopcount}$.

The third process in the system is the routing process at reach router. The routing process at router rcv reacts to incoming advertisements and updates the routing table entry, $(\mathsf{hops}(\mathsf{rcv}), \mathsf{nextN}(\mathsf{rcv}), \mathsf{nextR}(\mathsf{rcv}))$, at rcv. Essentially, if an advertisement, $(\mathsf{src}, \mathsf{hopcount})$, arriving at rcv through net, is such that $\mathsf{hopcount} + 1 < \mathsf{hops}(\mathsf{rcv})$ or $\mathsf{src} = \mathsf{nextR}(\mathsf{rcv}) \wedge \mathsf{net} = \mathsf{nextN}(\mathsf{rcv})$, then the routing table at rcv is updated so that $\mathsf{hops}(\mathsf{rcv}) = \mathsf{hopcount} + 1$ and $\mathsf{nextN}(\mathsf{rcv}) = \mathsf{net}$ and $\mathsf{nextR}(\mathsf{rcv}) = \mathsf{src}$. In HOL, we represent this process by a state update function, $\mathsf{update} : \mathsf{rtable} \to ('\mathsf{router} * '\mathsf{network} * '\mathsf{router} * \mathsf{num}) \to \mathsf{rtable}$, which, given a protocol state, $(\mathsf{hops}, \mathsf{nextN}, \mathsf{nextR})$, and an advertisement event $(\mathsf{src}, \mathsf{net}, \mathsf{rcv}, \mathsf{hopcount})$, computes the new protocol state. The HOL code for the update function is given for illustration in Appendix A.3.

Trace Semantics The observable behavior of the network and advertising processes is essentially an infinite sequence of advertisement events. Therefore, we choose to express the semantics of these processes as an event trace—an infinite sequence of tuples $(\mathsf{src}_i, \mathsf{net}_i, \mathsf{rcv}_i)$, representing advertisement events. Such a trace is considered *valid* only if

- the trace is fair—$\forall r_1, r_2 : \text{'}\mathsf{router}.\forall i.\exists j > i.(\mathsf{src}_i = r_1)\ \wedge\ (\mathsf{rcv}_i = r_2)$, and
- the events are possible—$\forall i.(\mathsf{conn}\ \mathsf{src}_i\ \mathsf{net}_i) \wedge (\mathsf{conn}\ \mathsf{rcv}_i\ \mathsf{net}_i.)$

The hopcount field of the advertisement can be filled in as follows: Suppose that at the i^{th} step (event) of the protocol, the state of the protocol is $s_i = (\mathsf{hops}, \mathsf{nextN}, \mathsf{nextR})$, then

- if $\mathsf{nextN}(\mathsf{src}_i) = \mathsf{net}_i$, then src_i sends an advertisement $(\mathsf{src}_i, 16)$ to rcv_i instantaneously via net_i;
- otherwise, src_i sends an advertisement $(\mathsf{src}_i, \mathsf{hops}(\mathsf{src}_i))$ to rcv_i instantaneously via net_i.

Given an event trace, the routing processes react to the events and update the protocol state. This produces an infinite state sequence of the protocol s_i defined as follows:

- s_0 is any sound state
- $s_{i+1} = \mathsf{update}\ s_i\ (\mathsf{src}_i, \mathsf{net}_i, \mathsf{rcv}_i, \mathsf{hopcount}_i)$, where the hopcount field is filled in as described above.

Thus the semantics of the update processes is the state sequence it can generate for a given event trace. All properties desired of the protocol are expressed and proved in terms of this state sequence. In particular, the convergence theorem states that, given any valid event trace, the states generated in the sequence must converge to the optimal routing table.

3.2 RIP in Promela

Promela [10, 17] is a natural specification language for network protocols. In addition to C-like programming constructs, it supports non-determinism, dynamic processes, and synchronous/asynchronous channel communication between processes. We translate the pseudo-code given in Appendices A.1, A.2 into Promela. A fragment of the resulting Promela code corresponding to the routing process is shown in Appendix A.4.

As in the HOL specification, at each router, we have a routing process and an advertisement process. The advertisements process is a simple non-terminating while loop that keeps sending advertisement to all its neighboring networks. The routing process waits for input advertisements and processes them as before. Finally we have a network process for each network, which simply implements the broadcast mechanism by taking advertisements sent to the network and transporting them to the input buffers of all the routers connected to it. It is only the network processes that know the topology of the network, the routing and advertisement processes only know the networks they are directly connected to.

Once all the above Promela processes are in place, we use SPIN to simulate the protocol for sample topologies to check our model. We can also verify that the protocol works for small topologies. A point worth noting is that in varying the topologies, all we need to change is the encoding of the network processes. The routing and advertising processes operate above this connection layer. In effect, the network processes can pretend to have an arbitrary topology and the routing/advertisement processes will not know the difference. We use this property later in our SPIN proofs of convergence, where we fool a solitary update process to believe it is part of a larger network.

4 Convergence of RIP

In this section we present a proof of convergence for RIP. We prove that, in the absence of topology changes, RIP will find shortest routes to the destination d, from every router inside the range of 15 hops.

4.1 Proof Results

On the outermost level, our proof uses induction on distance from the destination. For each router r, *distance* to d is defined as

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

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

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

The key notion in our proof is that of the k-stability:

Definition 2 (Stability). *For $k \geq 1$, we say that the universe is k-*stable *if both of the following properties hold:*

(S1) *Every router from the k-circle has its metric set to the actual distance to d. Moreover, if r is not connected to d, it has its* nextR *set to the first router on some shortest path to d:*

$$\forall r.\ r \in C_k \ \Rightarrow \mathsf{hops}(r) = D(r)\ \wedge (\neg \mathsf{conn}\ r\ d\ \Rightarrow\ D(\mathsf{nextR}(r)) = D(r) - 1)$$

(S2) *Every router outside the k-circle has its* hops *strictly greater than k:*

$$\forall r.\ r \notin C_k\ \Rightarrow\ \mathsf{hops}(r) > k.$$

Our goal is to prove that under any fair advertisement trace, the universe is guaranteed to become k-stable, for every $k < 16$. This proof will be carried out by induction on k.

Recall our stipulation that the universe starts in a sound state. It is easy to show that sound states are 1-stable, so this gives us the basis of induction:

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

A key property of k-stability is that once it is achieved, it is preserved forever.This would not be true if our definition of stability did not contain condition S2. This condition strengthens the induction hypothesis enough that we can induct on k-stability.

Lemma 2 (Preservation of stability). *For any $k < 16$, if the universe $\mathcal{U}$ is k-stable at some point, then it remains k-stable after an arbitrary number of advertisements.* □

Lemma 1 is easily proved in HOL using the definition of soundness. We also prove Lemma 2 in HOL, and it involves a significantly larger case analysis.

Progress from k-stability to $(k + 1)$-stability happens gradually—more and more routers start to conform to the conditions S1 and S2. This is why we need an additional, more refined, definition of stability which captures individual routers, rather than entire 'circles' of routers.

Definition 3. *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:*

$$\mathsf{hops}(r) = D(r) = k + 1 \quad \wedge \quad \mathsf{nextR}(r) \in C_k.$$

To prove that a k-stable universe eventually becomes $(k+1)$-stable, it suffices to show that every *router* at distance $(k + 1)$ eventually becomes $(k + 1)$-stable. This is the statement of the following lemma:

Lemma 3. *For any $k < 15$, and any router r such that $D(r) = k + 1$, if the universe $\mathcal{U}$ is k-stable at some point and the advertisement trace is fair, then r will eventually become $(k + 1)$-stable. Moreover, r then remains $(k + 1)$-stable indefinitely.* □

One of the key facts used in the proof of this lemma is fairness of the advertisement trace. Without fairness, neighbors of r would be allowed to simply stop advertising to r at any point. This would keep r's routing table unchanged and hence prevent it from ever achieving $(k + 1)$-stability.

Observe that Lemma 3 only involves one router r at a distance $k + 1$ from d. Starting from a k-stable state, we need to show that r converges to the correct value. Moreover, since all future states of the system are guaranteed to be k-stable (Lemma 2), r will receive advertisements from only two kinds of neighbors—those within the k-circle, and those outside it. This leads us to a finitary abstraction of the system. We can then prove the lemma using SPIN, which performs an exhaustive state search to prove that r will converge to the right value.

However, we need to prove that the finitary abstraction is property-preserving. This proof is done in HOL, by induction on the length of fair advertisement traces. The abstraction proof is the crucial link that allows us to join the HOL and SPIN results, without loss of rigor. The abstraction proof itself is rather complex with a large case analysis. However, the effort is justified since the finitary abstraction can then be used for multiple proofs with minor modifications. This re-use can be seen in the proof statistics in Section 6 (Table 1). The abstraction proof is represented in the table as the HOL portion of the second proof of Stability Preservation (Lemma 2).

Finally, using the fact that there are only finitely many routers, we easily derive the Progress Lemma which proves our inductive step:

Lemma 4 (Progress). *For any $k < 15$, if the universe $\mathcal{U}$ is k-stable at some point, then $\mathcal{U}$ will eventually become and remain $(k+1)$-stable indefinitely.* □

The main result about the convergence of RIP is a corollary of the above inductive argument:

Theorem 1 (Convergence of RIP). *Starting from an arbitrary sound initial state, evolving under an arbitrary fair advertisement trace, the universe $\mathcal{U}$ eventually becomes and remains 15-stable.* □

4.2 Significance of the Results

Our proof, which we call the *radius proof,* differs from the one described in [1] 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 stable region around d. The proof has three attributes of interest:

1. *It states a property about the RIP protocol, rather the asynchronous distributed 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. It also implicitly estimates the number of advertisements needed to progress from k-stability to $(k+1)$-stability. We exploit this in the next section to show a realtime bound on convergence.
3. *It uses a combination of theorem proving and model checking.* HOL is more expressive and serves as the main platform. SPIN is used to treat large case analyses.

5 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 certain protocol events happen. In the case of RIP, we use a single reliability assumption that describes the frequency of advertisements.

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 advertisement from each of its neighbors. □

RIP routers normally try to advertise every 30 seconds. However, because of congestion or some other condition, some packets may not go through. This is why the standard prescribes that a failure to receive an advertisement within 180 seconds is treated as a link failure. Thus, $\Delta = 180$ seconds satisfies the Fundamental Timing Assumption for RIP. Notice that the assumption implies fairness of the advertisement trace.

As before, we concentrate on a particular destination d. Our timing analysis is based on the notion of weak k-stability.

Definition 4 (Weak stability). *For* $2 \leq k \leq 15$, *we say that the universe* $\mathcal{U}$ *is* weakly k-stable *if all of the following conditions hold:*

(WS1) $\mathcal{U}$ *is* $(k-1)$*-stable.*
(WS2) $\forall r.\ D(r) = k \ \Rightarrow\ (r \text{ is } k\text{--}stable\ \vee\ \mathsf{hops}(r) > k)$.
(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 second disjunct in WS2 is what distinguishes it from the ordinary k-stability. Similarly as before, we have the preservation lemma:

Lemma 5 (Preservation of weak stability). *For any* $2 \leq k \leq 15$, *if the universe* $\mathcal{U}$ *is weakly* k*-stable at some time* t, *then it is weakly* k*-stable at any time* $t' \geq t$. □

This lemma and all of the subsequent results in this section are stated using real time. This is possible because of the Fundamental Timing Assumption, which provides a connection between discrete advertisement events and continuous time. Precisely, to show that some property P holds after a Δ time interval, it is enough to prove that P holds after each router receives at least one advertisement from each of its neighbors.

Now we show that the initial state inevitably becomes weakly 2-stable after RIP packets have been exchanged between every pair of neighbors:

Lemma 6 (Initial progress). *If the universe* $\mathcal{U}$ *is in a sound state and the topology does not change,* $\mathcal{U}$ *becomes weakly 2-stable after* Δ *time.* □

The main progress property says that it takes one Δ-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 7. *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* $t + \Delta$. □

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

Lemma 8 (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* $t + \Delta$. □

Lemmas 5, 6, and 8 are proved in SPIN (Lemma 7 is contained in Lemma 8). The technique for doing the proofs in SPIN is the same as in the previous section. We find a finitary abstraction of the system starting from the time when the universe if weakly k-stable. This abstraction allows us to prove the Lemmas in SPIN for a single router.

The *radius* of the universe (around 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 2 (RIP convergence time). *A sound 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 and is proved in HOL. 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 6)
after $2 \cdot \Delta$	weakly 3-stable	(by Lemma 8)
after $3 \cdot \Delta$	weakly 4-stable	(by Lemma 8)
...	...	...
after $(R-1) \cdot \Delta$	weakly R-stable	(by Lemma 8)
after $R \cdot \Delta$	R-stable	(by Lemma 7)

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 k-stability to $(k+1)$-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 2 shows an example that consists of k routers and has 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 sharp.

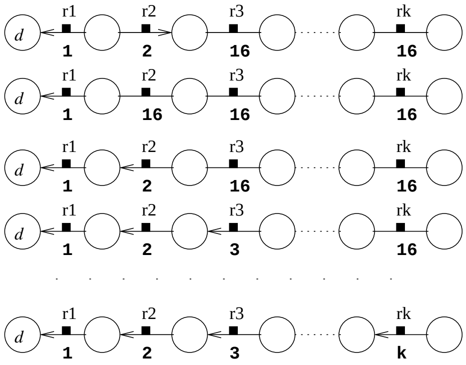

Fig. 2. Maximum Convergence Time

6 Analysis of Methodology

SPIN is extremely helpful for proving properties such as Lemma 8, which involve tedious case analysis. To illustrate this, assuming weak k-stability at time t, consider what it takes to show that condition WS2 for weak $(k+1)$-stability holds after Δ time. (WS1 will hold because of Lemma 7, 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 different kinds of 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. SPIN considered more cases than a manual proof would have required, 21,804 of them altogether for Lemma 8, 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.

Table 1 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

Table 1. Protocol Verification Effort on RIP Convergence

Task	HOL	SPIN
Modeling RIP	495 lines, 19 defs, 20 lemmas	141 lines
Stability Preservation Once	9 lemmas, 119 cases, 903 steps	
Stability Preservation Again	29 lemmas, 102 cases, 565 steps	207 lines, 439 states
Stability Progress	Reuse Stability Preservation	285 lines, 7116 states
Weak Stability Preservation	Reuse Stability Preservation	216 lines, 1019 states
Initial Weak Stability	Reuse Stability Preservation	221 lines, 1139 states
Weak Stability Progress	Reuse Stability Preservation	342 lines, 21804 states

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. In our investigations we have found that SPIN is generally memory bound, that is, it runs out of memory in a relatively short period of time if the state space it must search is too large. For our final RIP proofs, however, 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, some required more. 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 2, the Stability Preservation Lemma, using HOL only and HOL together with SPIN.

It is important to observe that the SPIN figures were derived from *final* runs. The typical process was as follows: attempt to prove a result with SPIN, find that it is too costly, apply an abstraction that was proved in HOL, and try the SPIN proof again on the abstracted problem (which presumably has a smaller set of cases to check). This was repeated until we were happy with the size of the SPIN state space and the clarity of the abstractions. This use of SPIN was worthwhile even if the proof was eventually carried out entirely in HOL since SPIN provided a quick way to 'debug' our lemmas. We experimented with the question of whether to stop with a mixed HOL/SPIN proof or complete the entire proof in HOL. A proof entirely in HOL arguably provides more confidence since the relationship between the HOL and SPIN parts of a proof are treated manually in our study. We proved stability preservation twice, once using HOL/SPIN and again using only HOL. Table 1 indicates some associated statistics showing that the complexity of the HOL proof dropped by about 40% at the cost of writing

207 lines of SPIN code. In future work we may attempt to measure programmer months since this would provide a more complete indication of scalability.

7 Related Work

Combining model checking with theorem proving has long been recognized as a very promising direction in effective formal methods [2]. There are primarily two ways in which the methodologies can be combined. Systems like PVS [16] use model checking as a decision procedure to solve finitary sub-goals in a deductive proof. On the other hand, model checking can be used to prove a finitary abstraction of a system where the soundness of the abstraction can be proved in a theorem prover [13, 14]. We use the latter methodology for our proofs—we carry out our induction and abstraction proofs in HOL90, while the induction step is proved for a finitary abstraction of the system in SPIN.

In recent years, a variety of protocol standards have been formally verified. Notable success has been achieved in verifying cache coherence protocols, bus protocols and endpoint communication protocols [2]. In the domain of routing protocols, there has been work on verifying ATM routing protocols [3], where the authors use SPIN to verify the absence of deadlock of the routing protocol for a few fixed configurations. An instance verification of an Active Network routing protocol has been carried out in Maude [18]. Formal testing support has been developed for multicast routing protocols [7]. Other work has been in the form of manual proofs of key safety properties [6, 15, 4, 1].

8 Conclusion

This paper provides the most extensive automated mathematical analysis of an internet routing protocol 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 proofs of the convergence of the RIP standard, and a sharp realtime bound for this convergence. We have also gained insight into strategies for combining a higher-order theorem prover such as HOL with a model checker such as SPIN in a unified methodology that leverages the expressiveness of the theorem prover and the high level of automation of the model checker to provide an efficient but high-confidence analysis.

Acknowledgments

We would like to thank the following people for their assistance and encouragement: Roch Guerin, Elsa L. Gunter, Luke Hornof, Sampath Kannan, and Insup Lee. This research was supported by NSF Contract CCR-9505469, DARPA Contract F30602-98-2-0198, and ARO Contract DAAG-98-1-0466.

References

1. Dimitri P. Bertsekas and Robert Gallager. *Data Networks.* Prentice Hall, 1991.
2. Edmund M. Clarke and Jeannette M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, December 1996. report by the Working Group on Formal Methods for the ACM Workshop on Strategic Directions in Computing Research.
3. D. Cypher, D. Lee, M. Martin-Villalba, C. Prins, and D. Su. Formal Specification, Verification, and Automatic Test Generation of ATM Routing Protocol: PNNI. In *Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE/PSTV) IFIP*, November 1998.
4. J.J. Garcia-Luna-Aceves and Shree Murthy. A Loop-Free Path-Finding Algorithm: Specification, Verification and Complexity. In *Proceedings of IEEE INFOCOM '95*, April 1995.
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. Timothy G. Griffin and Gordon Wilfong. An analysis of BGP convergence properties. In Guru Parulkar and Jonathan S. Turner, editors, *Proceedings of ACM SIGCOMM '99 Conference*, pages 277–288, Boston, August 1999.
7. Ahmed Helmy, Deborah Estrin, and Sandep Gupta. Fault-oriented Test Generation for Multicast Routing Protocol Design. In *Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE/PSTV) IFIP*, November 1998.
8. C. Hendrick. Routing information protocol. RFC 1058, IETF, June 1988.
9. Home page for the HOL interactive theorem proving system. `http://www.cl.cam.ac.uk/Research/HVG/HOL`.
10. Gerard J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1991.
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. Abdel Mokkedem, Ravi Hosabettu, Michael D. Jones, and Ganesh Gopalakrishnan. Formalization and Analysis of a Solution to the PCI 2.1 Bus Transaction Ordering Problem. *Formal Methods in System Design*, 16(1):93–119, January 2000.
14. Olaf Müller and Tobias Nipkow. Combining model checking and deduction for i/o-automata. In *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, May 1995.
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. N. Shankar. PVS: Combining specification, proof checking, and model checking. In Mandayam Srivas and Albert Camilleri, editors, *Formal Methods in Computer-Aided Design (FMCAD '96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 257–264, Palo Alto, CA, November 1996. Springer-Verlag.
17. Home page for the SPIN model checker. `http://netlib.bell-labs.com/netlib/spin/whatispin.html`.
18. Bow-Yaw Wang, José Meseguer, and Carl A. Gunter. Specification and formal verification of a PLAN algorithm in Maude. In *Proceedings of the International workshop on Distributed System Valdiation and Verification*, pages E:49–E:56. IEEE Computer Society Press, April 2000.

A Code Samples

A.1 Pseudocode for RIP Declarations

```
process RIPRouter
state:
  me                          // ID of the router
  interfaces                  // Set of router's interfaces
  known                       // Set of destinations with known routes
  hops_dest                   // Distance estimate
  nextR_dest                  // Next router on the way to dest
  nextIface_dest              // Interface over which the route advertisement was received
  timer expire_dest           // Expiration timer for the route
  timer garbageCollect_dest   // Garbage collection timer for the route
  timer advertise             // Timer for periodic advertisements

events:
  receive RIP (router, dest, hopCnt) over iface
  timeout (expire_dest)
  timeout (garbageCollect_dest)
  timeout (advertise)

utility functions:
  broadcast(msg, iface)
  { Broadcast message msg to all the routers attached to the network on the other side
    of interface iface.
  }
```

A.2 Pseudocode for RIP Event Handlers

```
event handlers:
  receive RIP (router, dest, hopCnt) over iface
  {
    newMetric ← min (1 + hopCnt, 16)
    if (dest ∉ known) then
    {
      if (newMetric < 16)
      {
        hops_dest ← newMetric
        nextR_dest ← router
        nextIface_dest ← iface
        set expire_dest to 180 seconds
        known ← known ∪ {dest}
      }
    } else
    { if (router = nextR_dest) or (newMetric < hops_dest)
      {
        hops_dest ← newMetric
        nextR_dest ← router
        nextIface_dest ← iface
        set expire_dest to 180 seconds
        if (newMetric = 16) then
        {
          set garbageCollect_dest to 120 seconds
        } else
        {
          deactivate garbageCollect_dest
  } } } }

  timeout (expire_dest)
  { hops_dest ← 16
    set garbageCollect_dest to 120 seconds
  }

  timeout (garbageCollect_dest)
```

$\{$ $known \leftarrow known - \{dest\}$
$\}$

timeout (*advertise*)
$\{$
for each $dest \in known$ **do**
for each $i \in interfaces$ **do**
$\{$
if $(i = nextIface_{dest})$ **then**
$\{$
broadcast $([RIP(me, dest, hops_{dest})],\ i)$
$\}$ **else**
$\{$
broadcast $([RIP(me, dest, 16)],\ i)$ // Split horizon with poisoned reverse
$\}$
$\}$
set *advertise* **to** *30 seconds*
$\}$

A.3 HOL Code for Update Function

```
val update_DEF = new_definition
   ("update",
    --`!(src:'router) (net:'network) (rcv:'router) (hopcount:num)
        (hops:'router->num) (nextN:'router->'network) (nextR:'router->'router).
            update (hops,nextN,nextR) (src,net,rcv,hopcount) =
            let (nh,nn,nr) =
               (((nextR(rcv)=src) /\ (nextN(rcv)=net)) =>
                          (SUC hopcount,net,src)
                       |  (((SUC hopcount) < hops(rcv)) =>
                                         (SUC hopcount,net,src)
                                      |  (hops(rcv),nextN(rcv),nextR(rcv))))
            in ((\r:'router.(r=rcv)=> nh | (hops(r))),
                (\r:'router.(r=rcv)=> nn | (nextN(r))),
                (\r:'router.(r=rcv)=> nr | (nextR(r))))`--);
```

A.4 Promela fragment for Routing Process

```
proctype Update(router ME){
         mesg adv;
         chan in = routerinput[ME];

         do
         ::  atomic{in?adv ->
                        if
                        :: (adv.src == rtable[ME].nextR) &&
                           (adv.net == rtable[ME].nextN) ->
                                if
                                :: adv.hopcount >= INFINITY ->
                                          rtable[ME].hops = INFINITY
                                :: adv.hopcount < INFINITY ->
                                          rtable[ME].hops = adv.hopcount + 1
                                fi
                        :: adv.hopcount + 1 < rtable[ME].hops ->
                                rtable[ME].nextR = adv.src;
                                rtable[ME].nextN = adv.net;
                                rtable[ME].hops = adv.hopcount + 1
                        :: else -> skip
                        fi}
         od
}
```