A mechanized proof of loop freedom of the (untimed) AODV routing protocol

Timothy Bourke[1,2], Rob van Glabbeek[3,4], and Peter Höfner[3,4]

[1] INRIA Paris-Rocquencourt, France
[2] Ecole normale supérieure, Paris, France
[3] NICTA, Australia
[4] Computer Science and Engineering, UNSW, Australia

Abstract. The Ad hoc On-demand Distance Vector (AODV) routing protocol allows the nodes in a Mobile Ad hoc Network (MANET) or a Wireless Mesh Network (WMN) to know where to forward data packets. Such a protocol is 'loop free' if it never leads to routing decisions that forward packets in circles. This paper describes the mechanization of an existing pen-and-paper proof of loop freedom of AODV in the interactive theorem prover Isabelle/HOL. The mechanization relies on a novel compositional approach for lifting invariants to networks of nodes. We exploit the mechanization to analyse several improvements of AODV and show that Isabelle/HOL can re-establish most proof obligations automatically and identify exactly the steps that are no longer valid.

1 Introduction

Mobile Ad hoc Networks (MANETs) and Wireless Mesh Networks (WMNs) are self-configuring wireless networks for mobile devices. Their nodes are reactive systems that cooperate to pass data packets from one node to another towards each packet's ultimate destination. This global service must satisfy certain correctness properties—for example, that data packets are never sent in circles. Proofs of such properties tend to be long and complicated, often involving many case distinctions over possible messages sent and combinations of Boolean predicates over internal data structures. For example, the only prior existing proof[1] of loop freedom of the Ad hoc On-demand Distance Vector (AODV) routing protocol—one of the four protocols currently standardized by the IETF MANET working group, and the basis of new WMN routing protocols such as HWMP in the IEEE 802.11s wireless mesh network standard [14]—is about 18 pages long and requires around 40 lemmas to prove the final statement [8]. This proof is based on a process-algebraic model.

Mechanizing process calculi and process-algebraic models in an Interactive Theorem Prover (ITP) like Isabelle/HOL [20] can now almost be considered routine [1, 9, 11, 13]. However, a lot of this work focuses on process calculi

[1] Earlier, and simpler, proofs appear in [2, 22] and [26], but none of them is complete and valid for AODV as standardized in [23]. We justify this statement in Section 9.

themselves—for example, by treating variable binding [1] or proving that bisimulation is a congruence [11,13]. While the study of security protocols has received some attention [7], comparatively little work has been done on mechanizing the application of such calculi to the practical verification of network protocols. In this paper, however, we focus on an application and mechanize the proof of loop freedom of AODV, a crucial correctness property. Our proof uses standard transition-system based techniques for showing safety properties [15,16], as well as a novel compositional technique for lifting properties from individual nodes to networks of nodes [4]. We demonstrate these techniques on an example of significant size and practical interest.

The development described in this paper builds directly on the aforementioned model and pen-and-paper proof of loop freedom of AODV [8]. While the process algebra model and the fine details of the original proof are already very formal, the implication that transfers statements about nodes to statements about networks involves coarser reasoning over execution sequences. Our mechanization simplifies and clarifies this aspect by explicitly stating the assumptions made of other nodes and by reformulating the original reasoning into an invariant form, that is, by reasoning over pairs of states rather than sequences of states.

Given that a proof already exists and that mechanization can be so time-consuming, why do we bother? Besides the added confidence and credibility that come with having even the smallest details fastidiously checked, the real advantage in encoding model, proof, and framework in the logic of an ITP is that they can then be analysed and manipulated (semi-)automatically. Section 8 describes how we exploited this fact to verify variations to the basic protocol, and Section 9 argues that such models aid review and repeatability. We expect that the work described will serve as a convenient and solid base for showing other properties of the AODV protocol and studying other protocols, and, eventually, to serve as a specification for refinement proofs. Finally, although any such work benefits from the accumulation of technical advances and engineering improvements to ITPs, we argue that it cannot yet be considered routine.

The paper is structured as follows. In Section 2 we informally describe the AODV protocol. Section 3 briefly states the theorem of loop freedom of AODV in the form given to Isabelle/HOL. The following three sections explain the meaning of this statement: Section 4 describes how the model of AODV in the process algebra AWN (Algebra for Wireless Networks) [8] is translated into Isabelle/HOL; Section 5 describes the formalization of network properties such as loop freedom; and, Section 6 explains our formalization of invariance of network properties. Section 7 summarizes how we proved the theorem in Isabelle/HOL. Section 8 describes several improvements of AODV, proposed in [8], and illustrates the use of Isabelle/HOL in proving loop freedom of these variants. Once the original proof has been mechanized, Isabelle/HOL can re-establish most proof obligations for these improvements automatically, and identify exactly the steps that are no longer valid and that need to be adjusted. A detailed discussion of related work follows in Section 9, followed by concluding remarks.

There is only space to show the most important parts of our mechanization of the process algebra AWN and of the model of AODV. Comparing these parts

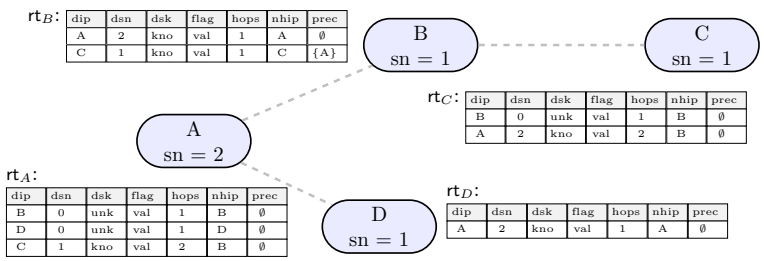

Fig. 1: Example AODV instance.

before [8, §§4–6] and after mechanization in Isabelle/HOL, should give sufficient clues about the remaining parts. By focusing mainly on the application (loop freedom of AODV), we only show a glimpse of our proof method. A companion paper [4] presents the technical details of the mechanization of AWN and the associated framework for compositional proof. Source files of the complete mechanization in Isabelle/HOL are available online [5].

2 The AODV routing protocol

The purpose of AODV [23] is to route data packets between nodes. Figure 1 shows an example network with four nodes addressed A, B, C, and D. Each node maintains a local *sequence number* (sn) and a *routing table* (rt). Imagine that the former are all set to 1, that the latter are all empty, and that A wants to send data to C. No route is known, so A increments its sn to 2 and broadcasts a Route Request (RREQ) message to its neighbours B and D. Both neighbours immediately add a routing table entry with the *destination address* (dip) as A, the *destination sequence number* (dsn) as 2, as sent by A, the *destination-sequence-number status* (dsk) as 'known' (kno), the *route status* (flag) as 'valid' (val), the *number of hops to the destination* (hops) as 1, the *next hop address* (nhip) as A, and an empty set of *precursors* (nodes known to be interested in this route).

Since neither B nor D has a routing table entry for C, they both in turn forward the RREQ message to their neighbours, which causes A to add entries for B and D, and C to add an entry for B. Since (forwarded) RREQs only include the sn of the originating node—A in this case—the dsn and dsk fields of these new entries are set to 0 and 'unknown' (unk), respectively. Node C also adds an entry for A to its routing table, with hop count 2 and next hop B. Since C is the destination, it replies to the request with a Route Reply (RREP) message, which is destined for A and unicast to B. On receipt, B updates its rt with a route to C (adding A as precursor) and forwards the message to A. When A receives this message it updates its rt and starts forwarding data packets to C via B. All established routing table entries are summarized in Figure 1.

Besides this basic scenario, AODV also supports early generation of RREP messages by 'intermediate nodes' [23, §6.6.2]: whenever an intermediate node has information about a route to the destination, rather than forwarding the

RREQ message, it generates an RREP message and sends it back to the originator of the RREQ message. AODV also supports Route Error (RERR) messages for invalidating routes (sent to the 'precursor nodes' associated with each entry).

AODV features various timing requirements to expire packets and entries, and to limit sending rates, as well as optional extensions, such as 'gratuitous RREPs' to improve bi-directional efficiency, and 'local repair' of routes on link loss. The model we mechanize includes the core functionality of AODV, but not timing details or optional features [8, §3].

3 Loop freedom

Routing protocols must continue to correctly route data even as nodes appear, disappear, and move. It is essential that they maintain *loop freedom*: the absence of cycles across different routing tables. For instance, the example network would have a cycle if D came into range of B, B updated its route for C to pass via D, and D added a route for C via A. Proofs of loop freedom when route replies are only generated by destination nodes are relatively subtle, but they become really delicate when intermediate nodes may also generate RREP messages.

The main result shown in our mechanization is:

Theorem 1 (AODV loop freedom). *For any well-formed network term* n*,* $\mathsf{closed}\ (\mathsf{pnet}\ (\lambda i.\ \mathsf{paodv}\ i\ \langle\langle\ \mathsf{qmsg})\ \mathsf{n}) \models \mathsf{netglobal}\ (\lambda\sigma.\ \forall\, \mathsf{dip}.\ \mathsf{irrefl}\ ((\mathsf{rt\text{-}graph}\ \sigma\ \mathsf{dip})^+))$.

Most of this paper is concerned with explaining the various elements of this statement and its proof in Isabelle/HOL. In sum, the variable n represents any well formed term describing a network instance—a term is well formed iff all nodes therein have distinct addresses. It is mapped to an automaton by the functions pnet and closed. A node with address i comprises an instance of the protocol, paodv i, reading messages from a queue process, qmsg. All reachable states of this model are shown to satisfy the formula at right of the '$\models$', which (1) maps a state structured like the network term into a function from each node address to that node's local state (netglobal), (2) abstracts this map into a directed graph with an arc from one node to another when the former has a valid routing table entry for a given dip to the latter (rt-graph), and finally, (3) claims that the transitive closure of each such graph is irreflexive.

4 Modelling AODV

In Isabelle/HOL we formalize AODV following the model from [8], which is expressed in a process algebra called AWN [8, §4]. In AWN, a network instance running a protocol is modelled in five layers, from the bottom up: (1) sequential processes, (2) local parallel composition at a single network node, (3) nodes of the form $ip{:}P{:}R$ with ip the node's address, R the set of reachable neighbours, and P the process running on the node, (4) partial networks of nodes, and, (5) networks closed to further interaction. The behaviour of each layer is defined by Structural Operational Semantics (SOS) rules over either process terms or lower layers. By including initial states, the first layer defines an automaton and

$\mathsf{p}_1 \oplus \mathsf{p}_2$ $\quad \mathsf{call(pn)}$ $\quad \{\!|\}[\![\mathsf{u}]\!]\,\mathsf{p}$ $\quad \{\!|\}\langle \mathsf{g} \rangle\,\mathsf{p}$ $\quad \{\!|\}\mathsf{unicast}(\mathsf{s}_{ip},\, \mathsf{s}_{msg}).\mathsf{p} \triangleright \mathsf{q}$ $\quad \{\!|\}\mathsf{broadcast}(\mathsf{s}_{msg}).\mathsf{p}$

$\{\!|\}\mathsf{groupcast}(\mathsf{s}_{ips},\, \mathsf{s}_{msg}).\mathsf{p}$ $\quad \{\!|\}\mathsf{send}(\mathsf{s}_{msg}).\mathsf{p}$ $\quad \{\!|\}\mathsf{receive}(\mathsf{u}_{msg}).\mathsf{p}$ $\quad \{\!|\}\mathsf{deliver}(\mathsf{s}_{data}).\mathsf{p}$

(a) Term constructors for ('s, 'p, 'l) seqp.

$$\frac{\xi' = \mathsf{u}\ \xi}{((\xi,\ \{\!|\}[\![\mathsf{u}]\!]\ \mathsf{p}),\ \tau,\ (\xi',\ \mathsf{p})) \in \mathsf{seqp\text{-}sos}\ \Gamma} \qquad \frac{((\xi,\ \mathsf{p}),\ \mathsf{a},\ (\xi',\ \mathsf{p}')) \in \mathsf{seqp\text{-}sos}\ \Gamma}{((\xi,\ \mathsf{p} \oplus \mathsf{q}),\ \mathsf{a},\ (\xi',\ \mathsf{p}')) \in \mathsf{seqp\text{-}sos}\ \Gamma}$$

$$\frac{((\xi,\ \Gamma\ \mathsf{pn}),\ \mathsf{a},\ (\xi',\ \mathsf{p}')) \in \mathsf{seqp\text{-}sos}\ \Gamma}{((\xi,\ \mathsf{call(pn)}),\ \mathsf{a},\ (\xi',\ \mathsf{p}')) \in \mathsf{seqp\text{-}sos}\ \Gamma} \qquad \frac{((\xi,\ \mathsf{q}),\ \mathsf{a},\ (\xi',\ \mathsf{q}')) \in \mathsf{seqp\text{-}sos}\ \Gamma}{((\xi,\ \mathsf{p} \oplus \mathsf{q}),\ \mathsf{a},\ (\xi',\ \mathsf{q}')) \in \mathsf{seqp\text{-}sos}\ \Gamma}$$

$$((\xi,\ \{\!|\}\mathsf{broadcast}(\mathsf{s}_{msg}).\mathsf{p}),\ \mathsf{broadcast}\ (\mathsf{s}_{msg}\ \xi),\ (\xi,\ \mathsf{p})) \in \mathsf{seqp\text{-}sos}\ \Gamma$$

(b) SOS rules for sequential processes: subset of seqp-sos.

Fig. 2: Sequential processes: terms and semantics

the others become functions over automata.

The four node network of Figure 1 is, for example, modelled as
closed (pnet (λi. paodv i ⟨⟨ qmsg) (⟨A; {B, D}⟩ ∥ (⟨B; {A, C}⟩ ∥ (⟨C; {B}⟩ ∥ ⟨D; {A}⟩))))), where the function closed models layer 5, closing a network, and pnet fabricates a partial network from a function mapping addresses to node processes and an expression describing the initial topology. For example, ⟨A; {B, D}⟩ becomes the node ⟨A : paodv A ⟨⟨ qmsg : {B, D}⟩ with address A, initial neighbours B and D, and running a local composition of the protocol process paodv (initialized with its address) fed by a queue qmsg of incoming messages. The communication ranges of nodes are independent of the structure of their composition and may change during an execution. We now briefly describe each layer in more detail. Full details of AWN, with all SOS rules, can be found in [8] and the source files [5].

(1) Sequential processes. Both the AODV protocol logic and the behaviour of message queues are specified by process terms of type ('s, 'p, 'l) seqp, parameterized by 's, the type of the data state manipulated by the term, 'p, the type of process names, and 'l, the type of labels. We write ξ or ξ' for variables of type 's, and p, p', q, or q' for those of type ('s, 'p, 'l) seqp. Labels are used to refer to particular control locations.

The term constructors are summarized in Figure 2a: *assignment*, $\{\!|\}[\![\mathsf{u}]\!]\,\mathsf{p}$, which transforms the data state deterministically (u has type 's ⇒ 's) and then acts as p; *guard/bind*, $\{\!|\}\langle \mathsf{g} \rangle\,\mathsf{p}$, which returns the set of states where the guard evaluates to true, one of which is chosen nondeterministically; *network synchronizations*, receive/unicast/broadcast/groupcast, whose destinations and contents depend on the data state; *internal communications*, send/receive/deliver, which do not need specified destinations and whose contents depend on the data state; *choice* (⊕), combining the possibilities of two subterms; and *call* (call), which jumps to a named process. The argument s_{msg} of broadcast is a data expression with variables, which evaluates to a message. It thus has type 's ⇒ msg. The argument u_{msg} of receive, on the other hand, is a variable that upon receipt of

a message is evaluated to the message received. It has the type of a message-dependent state change (msg ⇒ 's ⇒ 's). A guard is of type 's ⇒ 's set. It can express both a construct that nondeterministically binds variables, giving a set of possible successor states, and one that returns a singleton set containing the current state provided it satisfies a given condition and the empty set otherwise.

The SOS rules for sequential processes, seqp-sos, define a set of transitions. A transition is a triple relating a source state, an action, and a destination state. The states of sequential processes pair data components of type 's with control terms of type ('s, 'p, 'l) seqp. A set of transitions is defined relative to a *(recursive) specification* Γ of type 'p ⇒ ('s, 'p, 'l) seqp, which maps process names to terms. Some of the rules are shown in Figure 2b.

These elements suffice to express the control logic of the AODV protocol. We introduce six mutually recursive processes whose names are shown in Figure 3. This figure shows the control structure of the specification Γ_{aodv}, which maps each name to a term. The main process is called PAodv. It can broadcast control packets or send data packets—the two descending subtrees at the very left—or on receiving a message descend into one of the other terms depending on the message content—the five-pronged choice leading to the other labels. All paths loop back to PAodv. The smallest subprocess, at bottom right, is defined as

$$
\begin{aligned}
\Gamma_{\mathsf{aodv}}\ \mathsf{PNewPkt} = \mathsf{labelled}\ \mathsf{PNewPkt}\ (&\\
&\langle \lambda\xi.\ \mathsf{if\ dip}\ \xi = \mathsf{ip}\ \xi\ \mathsf{then}\ \{\xi\}\ \mathsf{else}\ \emptyset\rangle\\
&\mathsf{deliver(data)}\ .\ [\![\mathsf{clear\text{-}locals}]\!]\ \ \mathsf{call(PAodv)}\\
\oplus\ &\langle \lambda\xi.\ \mathsf{if\ dip}\ \xi \neq \mathsf{ip}\ \xi\ \mathsf{then}\ \{\xi\}\ \mathsf{else}\ \emptyset\rangle\\
&[\![\lambda\xi.\ \xi(\!|\mathsf{store} := \mathsf{add\ (data}\ \xi)\ (\mathsf{dip}\ \xi)\ (\mathsf{store}\ \xi)|\!)]\!]\\
&[\![\mathsf{clear\text{-}locals}]\!]\ \ \mathsf{call(PAodv)}\ ).
\end{aligned}
$$

It branches on whether or not the dip and ip variables have the same value in the current state, and then either delivers a message or updates the variable store. Each branch then loops back to the main process. The labelled function recursively labels the control states from PNewPkt-:0 through PNewPkt-:4.

The graph of Figure 3 summarizes the just over 100 control locations and shows that the model contains both significant branching and sequencing, some of which is exploited in the verification. The thicker, solid lines are synchronizing actions. The dashed lines are assignments or guards. Each straight sequence of dashed lines, which correspond to a sequence of assignments, could in fact be replaced by a single dashed line, by nesting state transformations. However,

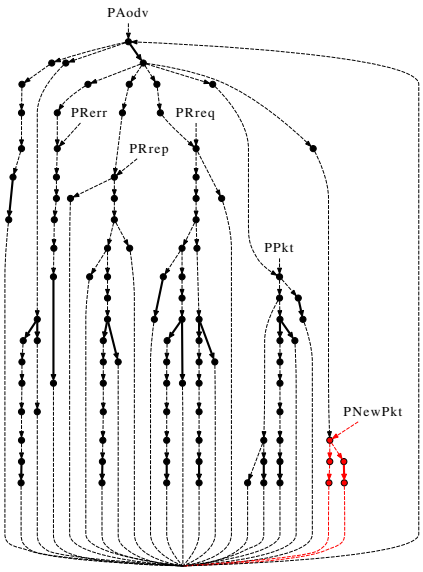

Fig. 3: Process term graph of Γ_{aodv}

this would make the model easier to get wrong, harder to read, and less like implementations in programming languages. Moreover, it is easier to verify many small steps, especially since typically most are dispatched automatically.

The AODV data state is modelled in a standard way using records [25, §3.3]. There are five global variables:[2] ip, the local address of type nat; sn, the current sequence number, also a nat; rreqs, a set of pairs of addresses and request identifiers that tracks already handled route requests; store, a partial map from addresses to a status and queue of pending data packets; and rt, the routing table, a partial map from addresses to entries of type sqn × k × f × nat × ip × ip set, where sqn and ip are synonyms for nat. Each subprocess also has its own local variables, which in the original process algebra [8] are often initialized from expressions during subprocess invocation. While it may seem 'tidy' to explicitly model variable locality, it complicates syntactic details and the aim of working ultimately on transition systems where there is no notion of a subprocess. Furthermore, the recursion through Γ_{aodv} induced by call already entails one or two technical details, as we discuss in [4], even before variable assignment is considered. So, rather than stay faithful to these details in the mechanization, we simply include the union of 12 local variables in the state record. When invoking a subprocess, these variables are set to arbitrary values,[3] by clear-locals, before combining assignment and call. This pragmatic solution works well in our setting.

The routing table, rt, is central to AODV and thus to this verification. Several functions are defined to access, add, update, and invalidate its entries. For example, the function nhop of type rt ⇒ ip ⇀ ip gives the next hop for an address if defined; update of type rt ⇒ ip ⇒ r ⇒ rt encodes the rules determining if and how an entry is modified, invalidate of type rt ⇒ (ip ⇀ sqn) ⇒ rt marks a set of routes as invalid (possibly individually setting sequence numbers), and addpreRT of type rt ⇒ ip ⇒ ip set ⇀ rt adds precursors to an entry. These (partial) functions are defined in the λ-calculus of Isabelle/HOL.

The process state is a pair of a data record and a process term. The (singleton) set of initial states is given by a function, σ_{aodv} i = {(aodv-init i, Γ_{aodv} PAodv)} where i is the initial value of ip. The process term represents the control state: a sort of a symbolic program counter. As the rules in Figure 2 indicate, process labels have no influence on the flow of control (unlike in [16]). They exist only to aid verification. The sets of initial states and transitions are bundled into a generic automaton structure with two fields to model an AODV process:

$$\mathsf{paodv\ i} = (\!|\mathsf{init} = \{(\mathsf{aodv\text{-}init\ i},\ \Gamma_{\mathsf{aodv}}\ \mathsf{PAodv})\},\ \mathsf{trans} = \mathsf{seqp\text{-}sos}\ \Gamma_{\mathsf{aodv}}|\!).$$

The states of this automaton (an overapproximation of the set of reachable states) are determined by the type of the sources and targets of the transitions.

Having considered sequential processes, we now consider the other four layers.

(2) Local parallel composition. An instance of AODV must always be willing to accept a message from its environment. To achieve this, and to model

[2] These variables are global in that their values are maintained indefinitely; however, they are local to each specific process running on a specific node.

[3] Almost: as sip may not equal ip, we use $\xi(\!|\mathsf{sip} := \mathsf{SOME\ x.\ x} \neq \mathsf{ip}\ \xi|\!)$.

asynchronous message transmission, the protocol process is combined with a simple FIFO-queue model (modelled as a sequential process): $\mathsf{paodv\ i} \langle\langle \mathsf{qmsg}$. The composition operator applies to automata:

$$\mathsf{s} \langle\langle \mathsf{t} = (\!|\mathsf{init} = \mathsf{init\ s} \times \mathsf{init\ t},\ \mathsf{trans} = \mathsf{parp\text{-}sos\ (trans\ s)\ (trans\ t)}|\!).$$

The rules for parp-sos can be found in [5, 8].

(3–4) Nodes and partial networks. Networks of AODV instances are specified as values of an inductive type: a net-tree is either a node $\langle \mathsf{i};\ \mathsf{R}_i \rangle$ with address i and a set of neighbour addresses R_i, or a composition of two net-trees $\mathsf{p}_1 \parallel \mathsf{p}_2$. The function pnet maps each such value to an automaton:

$$\begin{aligned}\mathsf{pnet\ np}\ \langle \mathsf{i};\ \mathsf{R}_i \rangle &= \langle \mathsf{i} : \mathsf{np\ i} : \mathsf{R}_i \rangle \\ \mathsf{pnet\ np}\ (\mathsf{p}_1 \parallel \mathsf{p}_2) &= (\!|\mathsf{init} = \{\mathsf{s}_1 \shortparallel \mathsf{s}_2 \mid \mathsf{s}_1 \in \mathsf{init\ (pnet\ np\ p}_1) \wedge \mathsf{s}_2 \in \mathsf{init\ (pnet\ np\ p}_2)\}, \\ &\qquad \mathsf{trans} = \mathsf{pnet\text{-}sos\ (trans\ (pnet\ np\ p}_1))\ \mathsf{(trans\ (pnet\ np\ p}_2))|\!),\end{aligned}$$

where np is a function from addresses i to parallel process expressions, such as λi. paodv i $\langle\!\langle$ qmsg, and where

$$\langle \mathsf{i} : \mathsf{np} : \mathsf{R}_i \rangle = (\!|\mathsf{init} = \{\mathsf{s}^{\mathsf{i}}_{\mathsf{R}_i} \mid \mathsf{s} \in \mathsf{init\ np}\},\ \mathsf{trans} = \mathsf{node\text{-}sos\ (trans\ np)}|\!).$$

The states of such automata mirror the structure of the original network term. Node states are denoted $\mathsf{s}^{\mathsf{i}}_{\mathsf{R}}$ and composed states are denoted $\mathsf{s}_1 \shortparallel \mathsf{s}_2$. During an execution of a network, the tree structure and addresses remain constant, but the neighbours, control states, and data states of nodes may evolve.

(5) Complete networks. Such a network is closed to new node interactions:

$$\mathsf{closed\ A} = \mathsf{A}(\!|\mathsf{trans} := \mathsf{cnet\text{-}sos\ (trans\ A)}|\!).$$

In sum, this section has presented the part of Theorem 1 to the left of the '$\models$'.

5 Stating network properties

Our verification exploits the inductive structure of net-trees and states. Experience taught us to keep this structure for as long as possible and only later to transform it into a partial mapping from addresses to data records, using:

$$\begin{aligned}\mathsf{netlift\ sr}\ (\mathsf{s}^{\mathsf{i}}_{\mathsf{R}}) &= [\mathsf{i} \mapsto \mathsf{fst\ (sr\ s)}] \\ \mathsf{netlift\ sr}\ (\mathsf{s} \shortparallel \mathsf{t}) &= \mathsf{netlift\ sr\ s} +\!\!+ \mathsf{netlift\ sr\ t},\end{aligned}$$

where sr divides the state into 'exposed' parts (the poadv data state) and 'masked' parts (the paodv control term and the states of qmsg). The fst then elides the latter. The result of netlift is a partial function from addresses to the exposed part of the state of the corresponding node. It is made total by mapping missing elements to the initial (data) state before being passed to a property P:

$$\mathsf{netglobal\ P} = \lambda\mathsf{s.\ P\ (default\ aodv\text{-}init\ (netlift\ fst\ s))},$$

where $\mathsf{default\ df\ f} = (\lambda\mathsf{i.\ case\ f\ i\ of\ None} \Rightarrow \mathsf{df\ i} \mid \mathsf{Some\ s} \Rightarrow \mathsf{s})$. P is a property over

a function, written σ, from addresses i to data records. It can thus speak, for instance, of the routing table maintained by node i. The function netglobal turns such a property into a property of network states s.

An example of P occurs in Theorem 1: $\lambda\sigma.\ \forall \mathsf{dip}.\ \mathsf{irrefl}((\mathsf{rt\text{-}graph}\,\sigma\,\mathsf{dip})^+)$. Here rt-graph is a function that, given an address dip and a function σ from addresses to data states, generates a routing graph: its vertices are all possible addresses i and there is an arc $(\mathsf{ip}, \mathsf{ip}')$ iff $\mathsf{ip} \neq \mathsf{dip}$ and the entry for dip in the routing table at node ip has the form $(*, *, \mathsf{val}, *, \mathsf{ip}', *)$. An arc in this routing graph indicates that ip' is the next hop on a valid route to dip known by ip; a path in a routing graph describes a route towards dip discovered by AODV. We say σ is *loop free* if the routing graphs $\mathsf{rt\text{-}graph}\,\sigma\,\mathsf{dip}$ are acyclic, for all destinations dip, i.e. if $P\sigma$ holds. A network state s is *loop free* iff the function $\mathsf{netmap}\,\mathsf{s}$ from addresses to data records is loop free, i.e. if $\mathsf{netglobal}\,P\,\mathsf{s}$. Finally, a routing protocol, such as AODV, is *loop free* iff all reachable network expressions are loop free. This quantification over reachable network expressions is encoded in the symbol $\models$.

In sum, this section has presented the part of Theorem 1 to the right of the '$\models$'.

6 Stating invariance of network properties

As Theorem 1 is a safety property, we need only consider questions of invariance, that is, properties of reachable states. The meta theory is classic [19, Part III].

Definition 1 (reachability). *For an automaton A and an assumption I over actions, reachable A I is the smallest set defined by the rules:*

$$\frac{s \in \mathit{init}\ A}{s \in \mathit{reachable}\ A\ I} \qquad \frac{s \in \mathit{reachable}\ A\ I \quad (s, a, s') \in \mathit{trans}\ A \quad I\ a}{s' \in \mathit{reachable}\ A\ I}$$

Definition 2 (invariance). *For an automaton A and an assumption I over actions, a predicate P is* invariant*, written $A \models (I \rightarrow) P$, iff $\forall s \in \mathit{reachable}\ A\ I.\ P\ s$.*

Definition 3 (transition invariance). *For an automaton A and an assumption I over actions, a predicate P is* transition invariant*, written $A \models\!\models (I \rightarrow) P$, iff $\forall a.\ I\ a \longrightarrow (\forall s \in \mathit{reachable}\ A\ I.\ \forall s'.\ (s, a, s') \in \mathit{trans}\ A \longrightarrow P\ (s, a, s'))$.*

We recover the standard definition when I is λ-. True and write simply $\mathsf{A} \models \mathsf{P}$.

In sum, this finishes the presentation of Theorem 1.

7 Our proof

To show invariance, we follow the compositional strategy elucidated in [24, §1.6.2]. That is, we take as a basic element automata with $\mathsf{seqp\text{-}sos}\ \Gamma_{\mathsf{aodv}}$ as the set of transitions, show invariance using induction, and then develop proof rules for each of the operators defined in the previous section to lift the results to complete networks. The inductive assertion method is also classic, see, for example, Manna & Pnueli [16, Rule INV-B]. Its core, in Isabelle, is the induction principle associated with Definition 1.

Both the original and mechanized proofs of Theorem 1 involve a succession of invariants and transition invariants leading to the ultimate result. As an example,

$$\mathsf{paodv\ i} \models \mathsf{onl}\ \Gamma_{\mathsf{aodv}}\ (\lambda(\xi, \text{-}).\ \forall\, \mathsf{ip} \in \mathsf{kD}\ (\mathsf{rt}\ \xi).\ 1 \leq \mathsf{the}\ (\mathsf{dhops}\ (\mathsf{rt}\ \xi)\ \mathsf{ip})) \qquad (1)$$

states that 'all routing table entries have a hop count greater than or equal to one' (kD gives the domain of a routing table; the the adds an obligation to show that ip is in the domain of dhops (rt ξ)). This particular predicate only ranges over the data state, ξ, but others also range over labels, for example,

$$\mathsf{paodv\ i} \models (\mathsf{recvmsg\ P} \rightarrow)\ \mathsf{onl}\ \Gamma_{\mathsf{aodv}}\ (\lambda(\xi, \mathsf{l}).\ \mathsf{l} \in \{\mathsf{PAodv\text{-}:1}\} \longrightarrow \mathsf{P}\ (\mathsf{msg}\ \xi)) \qquad (2)$$

states that 'if for every receive m, m satisfies P, then the msg variable also satisfies P at location PAodv-:1'. The map onl Γ P, defined by $\lambda(\xi, \mathsf{p}).\ \forall\, \mathsf{l} \in \mathsf{labels}\ \Gamma\ \mathsf{p}.\ \mathsf{P}\ (\xi, \mathsf{l})$, extracts labels from control states, which obviates the need to include and maintain process terms in invariants.[4]

Invariants like these are solved by establishing them for all possible initial states, and showing that they are preserved by all transitions. The soundness of this method is also formally justified in Isabelle/HOL [4].

This approach suffices for showing nearly all intermediate invariants, but not for expressing the final invariant from which Theorem 1 follows. The authors of the original proof [8, Theorem 7.30] introduce a notion of 'quality' of routing table entries, and show that it strictly increases along any valid route to a destination *dip*. They formalize this as

$$\text{“}dip \in \mathtt{vD}_N^{ip} \cap \mathtt{vD}_N^{nhip} \ \wedge\ nhip \neq dip\ \Rightarrow\ \xi_N^{ip}(\mathtt{rt}) \sqsubset_{dip} \xi_N^{nhip}(\mathtt{rt})\text{”},$$

where N is a "reachable network state", $\mathtt{vD}_N^{ip}$ are the addresses for which *ip* has valid routing entries, and *nhip* is the address of the next hop toward *dip* at *ip*. But our basic invariants, like (1) or (2), can only refer to the local model's state (ξ). How can we compare the states at two nodes (ξ^{ip} and ξ^{nhip}) without immediately introducing the whole model before the '$\models$'?

Our solution is to introduce 'open' versions of the SOS rules and operators, and of reachability and (transition) invariance.

The *open* SOS of AWN differs from the default (*closed*) version by modelling data states as (total) functions from node addresses to data records. That is, rather than define rules over a variable ξ of type state, they are defined over a variable σ of type ip $\Rightarrow$ state. At the level of sequential processes, the open equivalent of seqp-sos (oseqp-sos) is additionally parameterized by an address i, and the SOS rules only constrain this ith component. At the level of local parallel compositions within a node, we simply inherit the data state from the left argument (the process aodv).[5] The lifting to node expressions is unproblematic. The composition of two partial networks effectively synchronizes the state mappings: the only transitions that can occur are those where, given a source state (σ), both components agree on a destination state (σ').

[4] Using labels is standard, see, for instance, [16, Chap. 1], or the 'assertion networks' of [24, §2.5.1]. Isabelle rapidly dispatches uninteresting cases.

[5] This suffices for our work, but a symmetric solution may be preferable.

We developed a framework for stating invariants over automata with open transitions. These invariants differ from those like (1) and (2) in that assumptions are not stated over incoming messages but rather over synchronized and interleaved transitions—that is, over communications with an environment and over the independent actions of the environment—and properties are stated over the entire state of the network. We show invariants at the level of a single process (paodv i), with the additional obligation of showing their preservation under all interleaving transitions that satisfy the stated assumption, and then 'lift' them to arbitrary closed networks by applying a succession of generic lemmas, one for each layer of AWN. The additional obligation is exploited as a hypothesis in the induction that lifts results over partial networks (that is, when only one side acts, the property remains invariant). The lifting rules require showing that a process satisfies the assumptions on synchronizations and interleavings made in the invariant statement. These assumptions must also be lifted for each layer; care is required to avoid circularity in such assumption-guarantee invariants.

The framework includes a generic 'transfer' lemma that infers from an invariant over an open model, a similar invariant over the corresponding closed model—in our case, the very model presented in Section 4. One need only show a relation between the np given to pnet (see page 8) and a corresponding onp and opnet of the open model. This transfer of results means that all of the definitions and lemmas associated with the open model are but a proof strategy: they are not needed to understand the statement of Theorem 1 and their soundness is guaranteed by Isabelle/HOL. The details of this proof strategy are given in [4].

Route quality. For completeness, we include the definition used for route quality. We write $(\mathsf{rt}_1 \sqsubset_i \mathsf{rt}_2) = (\mathsf{rt}_1 \sqsubseteq_i \mathsf{rt}_2 \wedge \neg\ \mathsf{rt}_2 \sqsubseteq_i \mathsf{rt}_1)$, to mean that the quality of the route to address i in route table rt_2 is strictly better than that in rt_1, where,

$$(\mathsf{rt}_1 \sqsubseteq_i \mathsf{rt}_2) = (\mathsf{nsqn}\ \mathsf{rt}_1\ \mathsf{i} < \mathsf{nsqn}\ \mathsf{rt}_2\ \mathsf{i} \vee (\mathsf{nsqn}\ \mathsf{rt}_1\ \mathsf{i} = \mathsf{nsqn}\ \mathsf{rt}_2\ \mathsf{i} \wedge \mathsf{the}\ (\mathsf{dhops}\ \mathsf{rt}_2\ \mathsf{i}) \leq \mathsf{the}\ (\mathsf{dhops}\ \mathsf{rt}_1\ \mathsf{i}))),$$

provided $\mathsf{i} \in \mathsf{kD}\ \mathsf{rt}_1$ and $\mathsf{i} \in \mathsf{kD}\ \mathsf{rt}_2$. The function dhops rt i yields the number of hops to i according to rt. We encode the notion of *net sequence numbers* from [8, §7.5]:

$$\mathsf{nsqn}\ \mathsf{rt}\ \mathsf{i} = (\mathsf{if}\ \mathsf{flag}\ \mathsf{rt}\ \mathsf{i} = \mathsf{Some}\ \mathsf{val} \vee \mathsf{sqn}\ \mathsf{rt}\ \mathsf{i} = 0\ \mathsf{then}\ \mathsf{sqn}\ \mathsf{rt}\ \mathsf{i}\ \mathsf{else}\ \mathsf{sqn}\ \mathsf{rt}\ \mathsf{i} - 1),$$

where flag states whether a route is valid (val) or invalid (inv) and sqn gives the stored sequence number.

Results. Our mechanization of the AODV model and the proof of loop freedom (not including the framework of [4]) involves 360 lemmas, of which 40 are invariants, with a proof text spanning 80 printed pages. The pen-and-paper proof [8] involves 40 lemmas over 18 pages. Many of the mechanized lemmas are of course trivial, for example, simplification rules for projections from routing tables.

The pen-and-paper proof is fastidious and we did not find any major errors: (1) type checking found a minor typo in the model, (2) one proof invoked an incorrect invariant requiring the addition and proof of a new invariant based on an existing one, (3) a minor flaw in another proof required the addition of a new invariant. Of course, this was not known beforehand! Nevertheless, our mechanized proof provides supplementary evidence for the stated property.

8 Analysing variants of AODV

A mechanized model and proof greatly facilitates the analysis of protocol variants, such as different interpretations of the informal text of a standard, or proposed improvements for future versions of a standard. Since such variants often only differ in minor details, most proofs stay the same or are adapted automatically. An ITP tries to 'replay' the original proof and, in case of a failure, it indicates those proof steps that are no longer valid. One can thus concentrate on important changes in the proof. This avoids the tedious, time-consuming, and error-prone manual chore of establishing which steps remain valid for each invariant, especially for long proofs. We support our claim by proving the loop freedom property of four variants of AODV.

(1) Skipping route request identifiers. AODV uses route request identifiers to uniquely identify RREQ messages. Since it has been shown [8, §10.1] that a combination of IP addresses and sequence numbers adequately serves the same purpose, the identifier can be dropped and the size of the RREQ messages reduced. This very minor modification only requires a change in the type of RREQ, and related propositions and lemmas. Implementing these changes in Isabelle/HOL only took several minutes—the invariant lemmas were re-proved automatically.

(2) Forwarding route replies. During route discovery, an RREP message is unicast back towards the originator of the triggering RREQ message. Every intermediate node on the selected route processes the RREP message and, in most cases, forwards it towards the originator. However, intermediate nodes must discard RREP messages from which they cannot distil any new routing information. As a consequence, the originator node will not receive a reply.[6]

An alternative is to require intermediate nodes to forward *all* RREP messages. This behaviour is modelled by deleting three lines of the original specification; including one choice operator ($\oplus$) and two guards, potentially invalidating the proofs of many invariants. To avoid the forwarding of outdated information, we change three lines in the specification (including two guards) to ensure that the best available information is always sent; see [8, §10.2] for details.

Of the 360 lemmas in the original proof, only 7 are no longer valid. Four of these are easily repaired: since some lines of the specification are deleted, the automatically generated labels change and references in the proofs must be adapted—a tedious, but routine find-and-replace chore. So, in fact only three invariants require non-trivial user interaction and a new proof—around three hours of manual effort. This corresponds with the pen-and-paper proof, which requires a single page [8, pp. 106–107] and a new invariant [8, Prop. 7.38].

(3) From groupcast to broadcast. Each routing table entry contains a set of precursors (Section 2), a set of the IP addresses of all nodes that are currently known to be potential users of the route, and that are located one hop further

[6] See `http://www.ietf.org/mail-archive/web/manet/current/msg05702.html`.

away from the destination. This information is recorded so that these nodes can be informed via RERR message if the route becomes invalid. However, precursor lists are incomplete: nodes not handling a route reply have no information about precursors for routes established while handling RREQ messages—see [8, §10.4] for examples. As a consequence, some nodes cannot be informed of a link break and will use a broken route, and data packets can be lost.

One solution is to abandon precursors and to replace groupcasts by broadcasts. The AODV specification is updated by dropping the precursor field of routing table entries, and making minor changes to related functions and function calls. All 7 occurrences of groupcast must also be replaced by an appropriate broadcast-statement; in one case this necessitates the introduction of a new guard. 16 assignments dealing with the generation and maintenance of precursor lists, and the calculation of groupcast destinations, are deleted.

Several changes ensued. (1) Around 30 definitions and lemmas about precursor lists are no longer needed. (2) About 75 lemmas and proofs then require adjustments for typing errors and references to deleted lemmas. (3) The labels in 6 invariants must be updated. (4) One invariant requires a careful adjustment: the removal of one case of a case distinction.

In sum, the specification changes broke many invariant proofs, but these were easily fixed in around three hours.

(4) Forwarding route requests. During route discovery, an RREQ message is dropped if a node (destination or intermediate) replies to the sender with an RREP message. This dropping of the RREQ message may inadvertently lead to non-optimal routes to the originator at nodes laying 'downstream' of the node that sent the reply [17].

One possible solution is to make nodes forward *all* RREQs that they have not handled earlier. The forwarded RREQ messages must be augmented with a Boolean flag to indicate that a reply has already been generated and sent. The specification of this variant differs in only eight lines from the original [8, §10.5].

As before, the proof is adapted in response to feedback from Isabelle/HOL. (1) 17 lemmas must now include the newly introduced flag. (2) The labels in 4 invariants must be updated. (3) Only one invariant proof required major changes.

9 Related work

Bhargavan et al. [2]. Bhargavan, Obradovic, and Gunter apply a mix of manual reasoning, interactive theorem proving, and model checking to a preliminary draft (version 2) of AODV and show that this early draft is not loop free. They also suggest three improvements, of which one has been incorporated into the standard [23], and present a proof of loop freedom for the modified version. A central role in this proof is played by an invariant stating that along a route either sequence numbers increase, or, when they stay constant, that the hop count decreases [2, Theorem 17]. However, this is only true for valid routes—an assumption that is not stated. Even were the assumption adopted, it is not clear

how the property can be shown using step-by-step reasoning which must treat the case of invalid routes that become valid again. Looking at the proofs in [2], it turns out that Lemma 20(1) of [2] is invalid. This failure is surprising, given that according to [2] Lemma 20 is automatically verified by SPIN. A possible explanation might be that this lemma is obviously valid for the version of AODV prior to the recommendations of [2].

Bhargavan et al. concede that they do not formally prove the abstractions they use for model checking [2, p.565], and it is not otherwise clear whether or how they ensure consistency with their ITP models. Besides the obvious question of soundness, these manual steps make it harder to reproduce the stated results. If any part of the model is changed, the automatic components can be rerun, but both the relations between them and any other manual reasoning must be carefully re-evaluated. In contrast, our development is a complete mechanization that is automatically validated by Isabelle/HOL in less than 15 minutes.

Zhou et al. [26]. Zhou, Yang, Zhang, and Wang model AODV as a set of finite traces—lists of events—defined inductively using a technique expounded by Paulson [21]. The model features 15 cases for adding a new event to an existing trace τ; most involve conditions on 'observation functions' that recurse over τ to 'recreate' the system state. Zhou et al. show the invariant proposed by Bhargavan et al. but with an explicit assumption on route activity. They do so using an ingenious but intricate lemma ('l65') that exploits the identification of states and histories to reason across periods of route invalidity.

Both our mechanization and the original pen-and-paper proof [8] were completed without access to the details of Zhou et al.'s work. When we were able to examine their model in detail, we were reassured to see that the two models largely agree on the reading of the standard. But there are two crucial differences. First, we model route replies by intermediate nodes and Zhou et al. do not. This feature is a core part of the AODV standard [23, §6.6.2] and quite subtle—even small deviations, such as a different reading of the standard, risk introducing loops [10]. While we think it would be possible to add such replies to their model without introducing loops, extending the proof would not be trivial and would likely require new invariants similar to those that we use. Second, Zhou et al. model some timing details and we do not. But they sidestep central issues. For example, according to the standard a route is expired in two steps: "the Lifetime field in the routing table plays [a] dual role—for a valid route it is the expiry time, and for an invalid route it is the deletion time" [23, §6.11]. Zhou et al. model the first step but not the second. Route deletion, however, is fundamental to a timed model since all route information is lost; it complicates even the basic lemma that local sequence numbers never decrease (their 'l63').

Apart from content, the two models also differ in style. While the model of Zhou et al. is event-based and declarative, ours is more operational—it states what an abstract implementation of the protocol does step-by-step. The difference is important for two reasons: validation against the standard [23] and refinement to an implementation. Arguably, such standards are often quite operational, perhaps because they are written by and for implementers. We expect

it to be easier to state and show some kind of simulation relation between the states and transitions of our model and those of an implementation (model).

As this discussion shows, in addition to guaranteeing soundness and facilitating repeatability and reuse, mechanized proofs aid detailed comparisons with related work (provided the proof scripts are available for study). Since an ITP checks the proofs, one can focus on comparing models and properties. Furthermore, rather than puzzle over details omitted or unclear from published accounts, one can look for answers in the mechanization. AODV is an interesting case study since at least two mechanical models exist, and the protocol is of industrial relevance, complicated enough to be interesting, but not so large as to pose too many engineering problems.

Mechanically verifying reactive systems. Apart from the process-algebraic work described in the introduction, several other approaches for verifying reactive systems have been mechanized, namely UNITY [12], I/O Automata [18], and TLA$^+$ [6]. The main difference with our approach is that they typically do not distinguish control and data states—specifications are essentially flat sets of transitions. The last two frameworks, in particular, have focused on the verification of practical protocols but not, to our knowledge, on the kind of routing protocol exemplified by AODV.

10 Conclusion

We have presented a mechanical proof of a model that corresponds to an interpretation of the current version of the AODV standard [23]; the fidelity of this model is argued in [8]. It includes route replies from intermediate nodes but not timing features. Such a mechanization does more than confirm the correctness of the existing pen-and-paper proof, it provides a computerized object that can be examined by others and serve as a foundation for analyses of variants and extensions to, and other properties of AODV. We believe that our mechanization of the process algebra AWN, and the general framework for compositionally proving safety properties is also applicable to the study of other protocols.

A number of interesting questions remain. (1) What is the best way to manage variant models in a proof assistant? (2) How suitable is such a model for showing refinements to more detailed implementation models? (3) Can we validate our model against a real AODV implementation as has been done for the Transmission Control Protocol [3]? (4) We have not modelled timing details, which is not just a question of modelling, but also one of which invariants are needed to show loop freedom when routes can be spontaneously deleted. Ideally, timing details could also be incorporated into refinement proofs.

Acknowledgements. The authors thank G. Klein and M. Pouzet for their support and complaisance, M. Daum for his participation in early work, and L. Mandel and anonymous reviewers for their comments on a previous version.

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

1. Bengtson, J., Parrow, J.: Psi-calculi in Isabelle. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs'09. LNCS, vol. 5674, pp. 99–114. Springer (2009)
2. Bhargavan, K., Obradovic, D., Gunter, C.A.: Formal verification of standards for distance vector routing protocols. J. ACM 49(4), 538–576 (2002)
3. Bishop, S., Fairbairn, M., Norrish, M., Sewell, P., Smith, M., Wansbrough, K.: Engineering with logic: HOL specification and symbolic-evaluation testing for TCP implementations. In: POPL'06. pp. 55–66. ACM (Jan 2006)
4. Bourke, T., van Glabbeek, R.J., Höfner, P.: Showing invariance compositionally for a process algebra for network protocols. In: Klein, G., Gamboa, R. (eds.) ITP'14. LNCS, vol. 8558, pp. 144–159. Springer (2014)
5. Bourke, T., Höfner, P.: Loop freedom of the (untimed) AODV routing protocol. Archive of Formal Proofs (2014), `http://afp.sf.net/entries/AODV.shtml`
6. Chaudhuri, K., Doligez, D., Lamport, L., Merz, S.: Verifying safety properties with the TLA$^+$ proof system. In: Giesl, J., Hähnle, R. (eds.) IJCAR'10. LNCS, vol. 6173, pp. 142–148. Springer (2010)
7. Dutertre, B., Schneider, S.: Using a PVS embedding of CSP to verify authentication protocols. In: Gunter, E.L., Felty, A.P. (eds.) TPHOLs'97. LNCS, vol. 1275, pp. 121–136. Springer (1997)
8. Fehnker, A., van Glabbeek, R.J., Höfner, P., McIver, A., Portmann, M., Tan, W.L.: A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. Technical Report 5513, NICTA (2013), `http://arxiv.org/abs/1312.7645`
9. Feliachi, A., Gaudel, M.C., Wolff, B.: Isabelle/Circus: A process specification and verification environment. In: Joshi, R., Müller, P., Podelski, A. (eds.) VSTTE'12. LNCS, vol. 7152, pp. 243–260. Springer (2012)
10. van Glabbeek, R.J., Höfner, P., Tan, W.L., Portmann, M.: Sequence numbers do not guarantee loop freedom —AODV can yield routing loops—. In: MSWiM'13. pp. 91–100. ACM (2013)
11. Göthel, T., Glesner, S.: An approach for machine-assisted verification of Timed CSP specifications. Innovations in Systems and Software Engineering 6(3), 181–193 (Sep 2010)
12. Heyd, B., Crégut, P.: A modular coding of UNITY in COQ. In: Goos, G., Hartmanis, J., Leeuwen, J., Wright, J., Grundy, J., Harrison, J. (eds.) TPHOLs'96, LNCS, vol. 1125, pp. 251–266. Springer (1996)
13. Hirschkoff, D.: A full formalisation of π-calculus theory in the Calculus of Constructions. In: Gunter, E.L., Felty, A.P. (eds.) TPHOLs'97. LNCS, vol. 1275, pp. 153–169. Springer (1997)
14. IEEE: IEEE standard for information technology—telecommunications and information exchange between systems—local and metropolitan area networks—specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 10: Mesh networking (2011)
15. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)
16. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems: Safety. Springer (1995)
17. Miskovic, S., Knightly, E.W.: Routing primitives for wireless mesh networks: Design, analysis and experiments. In: INFOCOM'10. pp. 2793–2801. IEEE (2010)

18. Müller, O.: I/O automata and beyond: Temporal logic and abstraction in Isabelle. In: Grundy, J., Newey, M.C. (eds.) TPHOLs'98. LNCS, vol. 1479, pp. 331–348. Springer (1998)
19. Müller, O.: A Verification Environment for I/O Automata Based on Formalized Meta-Theory. Ph.D. thesis, TU München (1998)
20. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
21. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. J. Computer Security 6(1–2), 85–128 (Sep 1998)
22. Perkins, C.E., Royer, E.M.: Ad-hoc On-Demand Distance Vector Routing. In: Mobile Computing Systems and Applications (WMCSA'99). pp. 90–100. IEEE (1999)
23. Perkins, C.E., Belding-Royer, E.M., Das, S.R.: Ad hoc on-demand distance vector (AODV) routing. RFC 3561 (Experimental), Network Working Group (2003), `http://www.ietf.org/rfc/rfc3561.txt`
24. de Roever, W.P., de Boer, F., Hannemann, U., Hooman, J., Lakhnech, Y., Poel, M., Zwiers, J.: Concurrency Verification: Introduction to Compositional and Noncompositional Methods. Cambridge Tracts in Theoretical Computer Science 54, Cambridge University Press (2001)
25. Schirmer, N., Wenzel, M.: State spaces—the locale way. In: Huuck, R., Klein, G., Schlich, B. (eds.) SSV'09. ENTCS, vol. 254, pp. 161–179. Springer (2009)
26. Zhou, M., Yang, H., Zhang, X., Wang, J.: The proof of AODV loop freedom. In: WCSP'09. IEEE (2009)