Formal Methods in System Design manuscript No.
(will be inserted by the editor)

Model Checking Mobile Ad Hoc Networks

Fatemeh Ghassemi · Wan Fokkink

the date of receipt and acceptance should be inserted later

Abstract Modeling arbitrary connectivity changes within mobile ad hoc networks (MANETs) makes application of automated formal verification challenging. We use constrained labeled transition systems as a semantic model to represent mobility. To model check MANET protocols with respect to the underlying topology and connectivity changes, we introduce a branching-time temporal logic. The path quantifiers are parameterized by multi-hop constraints over topologies, to discriminate the paths over which the temporal behavior should be investigated; the paths that violate the multi-hop constraints are not considered. A model checking algorithm is presented to verify MANETs that allow arbitrary mobility, under the assumption of reliable communication. It is applied to analyze a leader election protocol.

1 Introduction

In mobile ad hoc networks (MANETs), nodes are equipped with wireless transceivers to communicate with each other. Wireless communication is restricted; only nodes located in the range of a transmitter receive data. Therefore, nodes rely on their neighbors to communicate with others along multi-hop connections. Due to e.g. noise from the environment, interferences, and temporary communication link errors, wireless communication is inherently unreliable, which together with mobility of nodes complicates the design of MANET protocols. Formal methods provide valuable tools to design, evaluate and verify MANET protocols.

Restricted Broadcast Process Theory (*RBPT*) [17] is a process algebra that targets the specification and verification of MANETs, taking into account mobility. *RBPT*

F. Ghassemi
University of Tehran, Iran
Tel.: +98-21-82084995
E-mail: fghassemi@ut.ac.ir

W. Fokkink
Vrije Universiteit Amsterdam, The Netherlands
E-mail: w.j.fokkink@vu.nl

specifies a MANET by composing nodes using a restricted local broadcast operator. A strong point of *RBPT* is that the underlying topology is not specified in the syntax, which would make it hard to set up the initial topology and maintain the evolving topology for each scenario in a verification. In related frameworks, mobility is modeled as an arbitrary manipulation of the underlying topology (given as part of the semantic state), which may make the model infinite and insusceptible to automated verification techniques; the number of topologies grows exponentially with respect to the number of nodes in the network. Instead, the semantic model of *RBPT* is a *constraint labeled transition system* (*CLTS*) [18], in which transitions are enriched with so-called network constraints, to symbolically express the set of possible topologies for which such a transition is possible. Network constraints are (dis)connectivity pairs like $\ell \rightsquigarrow \ell'$ or $\ell \not\rightsquigarrow \ell'$, denoting whether or not the node with logical address ℓ' is located in the range of the node with logical address ℓ, and consequently can receive data from the latter node. This symbolic representation of network topologies in the semantics results in a compact semantic model, which facilitates the investigation of families of properties by existing automated verification techniques. In this paper, CLTSs are generated through the algebraic framework from [17], which allows for equational reasoning.

Properties of MANETs tend to be weaker than of wired networks. This is due to the topology-dependent behavior of communication, and consequently the need for multi-hop communication between nodes. For instance, the important property of *packet delivery* in routing or information dissemination protocols in the context of MANETs becomes: if there exists an end-to-end route between two nodes A and B *for a sufficiently long period of time*, then packets sent by A will eventually be received by B [13]. To verify such properties using an existing temporal logic such as CTL [7], a preprocessing step is required to enrich states with propositions that indicate restrictions over the underlying topology. The large number of topologies and mobility scenarios leads to an exponential blow-up if they are considered explicitly and individually. In [12] this bottleneck was circumvented by considering only specific mobility scenarios.

To remove efforts prior to verification, improve memory usage, gain efficiency, and extend verification to arbitrary mobility changes, we introduce the temporal logic *Constrained Action Computation Tree Logic* (CACTL), which is interpreted over CLTSs. It extends ACTLW [25] (which in turn is based on Action CTL [11]) with topological constraints. In CACTL, the path quantifier operator *All* is parameterized by a constraint on the underlying topologies, so that only paths are considered for which the constraint is satisfied, for example that a multi-hop connection is present. This extension can in principle be employed with regard to any temporal logic.

The new dimension of topology restrictions in the logic requires that the model checking algorithm is modified appropriately. We present a model checking algorithm for CACTL, implemented in Java. This approach is flexible and efficient for the specification and verification of mobile behavior, compared to existing approaches. As an example we show how properties of a leader election protocol for MANETs can be verified.

The aim of the current paper is to detect malfunctions of a MANET protocol caused by conceptual mistakes in the protocol design, rather than by unreliable communication. Therefore, we consider reliable communication here, and refer to the process algebra as *Reliable RBPT (RRBPT)*. The main difference between *RRBPT* and *RBPT* is in their operational rules: in *RRBPT*, nodes lose a communication only when they are disconnected. Furthermore, *RRBPT* introduces a new operator to examine the status of a link, and upon its (non-)existence adapts its behavior accordingly to successfully accomplish its tasks. This operator abstracts away from the neighbor discovery service implemented at the network layer, by which a node becomes aware of its connection topology discovery. The neighbor discovery protocol is implemented by periodically sending *hello* messages and acknowledging such messages received from a neighbor. The novel *sensing* operator, inspired by the work in [23], obviates the explicit modeling of this service, and consequently makes modeling of protocols using such a service easier. The neighbor discovery service is modeled implicitly in the semantics in [23]. To this aim, an arbitrary subset of a node's neighbors is considered as the neighbors discovered by such a protocol.

A preliminary version of the current paper appeared as [16]. In comparison to [16], some fundamental changes were made to the syntax as well as the semantics of CACTL, as explained at the end of Section 5. Also the model checking algorithm for CACTL was overhauled and made more efficient. A third major development is the implementation of a more mature model checker.

The paper is structured as follows. Section 2 provides an overview on existing approaches for the verification of MANETs. Section 3 introduces CLTSs and explains how to implicitly model mobility of nodes. Section 4 describes the process theory *RRBPT*. Section 5 introduces the syntax and semantics of CACTL. Section 6 provides a model checking algorithm and discusses its implementation. Section 7 illustrates the expressiveness of CACTL in the analysis of MANET protocols. Section 8 concludes the paper.

2 Related Work

MANET protocols have been studied either using existing formal specification languages and verification tools such as SPIN [30,34,3] or UPPAAL [34,35,24,12], or by introducing a specific formal framework, mainly with an algebraic approach. Important modeling challenges in MANETs are *local broadcast*, *underlying topology* and *mobility*. The modeling approach using existing formalisms can be summarized as follows: The underlying topology is modeled by a two-dimensional array of Booleans, mobility by explicit manipulation of this matrix, and local broadcast by unicasting to all nodes with whom the sending node is presently connected, using the connectivity matrix. Verification tends to be based on model checking techniques restricted to a pre-specified mobility scenario. Lack of support for compositional modeling and arbitrary topology changes has motivated new approaches with a primitive for local broadcast and support of arbitrary mobility. These approaches include CBS#, bKlaim, CWS, CMAN, CMN, ω-calculus, *RBPT*, CSDT, and AWN [28,29,27,20,26, 31,17,19,23,13]. The common point among them (except *RBPT*) is implicit manipulation of the underlying topology in the semantics to model arbitrary connectivity changes and mobility. The analysis techniques supported by these frameworks, except bKlaim and AWN, are based on a behavioral congruence relation. In [18] we provided an axiomatization to derive that a specification of a MANET protocol is observably equal to a specification of its desired external behavior. Equational reasoning (applied at the syntactic or the semantic level) requires either abstraction from the actual specification of the MANET protocol, or knowledge about the overall behavior of the MANET beforehand.

The mix of broadcast behavior and mobility in MANETs leads to state-space explosion, hampering the application of automated verification techniques like model checking. In bKlaim [29], the semantic model is abstracted to a finite labeled transition system such that the mobility information is preserved; a variant of ACTL is introduced to determine which properties hold if movement of nodes is restricted. To this aim, ACTL operators are parameterized by a set of possible network configurations (topology). However, topology-dependent behavior cannot be checked. AWN [13] verifies topology-dependent behavior properties using CTL [7], by treating a transition label carrying (dis)connectivity information as a predicate of its succeeding state and defining predicates over the topology as part of the syntax. This approach can be extended to algebras, e.g. CMAN and ω-calculus, with (dis)connectivity information on transition labels. However, this approach needs auxiliary strategies to extract predicates from the states and transitions, to restrict connectivity changes during model checking and thus limit the state space. These challenges are tackled with the help of the model checker UPPAAL, by transforming AWN specifications to automata and exploiting an auxiliary automaton which statically restricts connectivity changes [12], similar to [24]. In [5] loop freedom of the AODV routing protocol for MANETs was proved using the interactive theorem prover Isabelle/HOL.

3 Preliminaries

We explain the semantic model of MANET protocols, how mobility is addressed, which information can be inferred from a semantic model, and finally how mobility can be restricted.

3.1 Constrained labeled transition systems

Communication in wireless networks tends to be based on local (also called restricted) broadcast: Only nodes that are located in the transmission area of a sender can receive messages from this sender. A node B is *directly connected to* a node A if B is located within the transmission range of A. This asymmetric connectivity relation between nodes introduces a topology concept. A *topology* is a function $\gamma : \mathit{Loc} \rightarrow I\!\!P(\mathit{Loc})$ with $\forall \ell \in \mathit{Loc}(\ell \not\in \gamma(\ell))$, where Loc denotes a finite set of (hardware) addresses A, B, C. The set Loc is extended with the unknown address ? to denote the address of a node which is still not known. For instance, the leader address of a node can be initialized to this value. Furthermore, to define the semantics of communicating nodes in terms of restrictions over the topology in a compositional way, the semantics of receiving actions can be defined through an unknown sender which will be replaced by a known address when they are with the corresponding sending actions at specific nodes (see Section 4.2).

*Constrained labeled transition system*s (CLTSs) [18] provide a semantic model for the operational behavior of MANETs. A transition label is a pair of an action and a *network constraint*, restricting the range of possible underlying topologies. A network constraint $\mathcal{C}$ is a set of connectivity pairs $\rightsquigarrow$: $Loc \times Loc$ and disconnectivity pairs $\not\rightsquigarrow$: $Loc \times Loc$. In this setting, non-existence of (dis)connectivity information between two addresses implies lack of information about this link (which can e.g. be helpful when the link has no effect on the evolution of the network). For instance, $B \rightsquigarrow A$ denotes that A is connected to B directly and consequently A can receive data sent by B (note that the direction of $\rightsquigarrow$ implies the direction of data flow), while $B \not\rightsquigarrow A$ denotes that A is not connected to B directly and consequently cannot receive any message from B. We write $\{B \rightsquigarrow A, C,\ B \not\rightsquigarrow D, E\}$ instead of $\{B \rightsquigarrow A,\ B \rightsquigarrow C,\ B \not\rightsquigarrow D,\ B \not\rightsquigarrow E\}$.

Let $\mathbb{C}(Loc)$ denote the set of network constraints that can be defined over network addresses in Loc. A network constraint $\mathcal{C}$ is said to be *well-formed* if $\nexists \ell, \ell' \in Loc\ (\ell \rightsquigarrow \ell' \in \mathcal{C} \land \ell \not\rightsquigarrow \ell' \in \mathcal{C})$. Let $\mathbb{C}^v(Loc) \subseteq \mathbb{C}(Loc)$ denote the set of well-formed network constraints that can be defined over network addresses in Loc. Each well-formed network constraint $\mathcal{C}$ represents the set of network topologies that satisfy the (dis)connectivity pairs in $\mathcal{C}$, i.e., $\{\gamma \mid \mathcal{C} \subseteq \mathcal{C}_\Gamma(\gamma)\}$, where $\mathcal{C}_\Gamma^+(\gamma) = \{\ell \rightsquigarrow \ell' \mid \ell' \in \gamma(\ell)\} \cup \{\ell \not\rightsquigarrow \ell' \mid \ell' \notin \gamma(\ell)\}$ extracts all one-hop (dis)connectivity information from γ. So the empty network constraint $\{\}$ denotes all possible topologies over Loc. The negation $\neg\,\mathcal{C}$ of network constraint $\mathcal{C}$ is obtained by negating all its (dis)connectivity pairs. Clearly, if $\mathcal{C}$ is well-formed then so is $\neg\,\mathcal{C}$.

Let Act_τ be the set of actions, including the silent action τ, ranged over by η. A *CLTS* is of the form by $\langle S, \Lambda, \rightarrow, s_0\rangle$, with S a set of states, $\Lambda \subseteq \mathbb{C}^v(Loc) \times Act_\tau$, $\rightarrow \subseteq S \times \Lambda \times S$ a transition relation, and $s_0 \in S$ the initial state. A transition $(s, (\mathcal{C}, \eta), s') \in \rightarrow$, denoted by $s \xrightarrow{(\mathcal{C},\eta)} s'$, expresses that a MANET protocol in state s with an underlying topology $\gamma \in \mathcal{C}$ can perform action η to evolve to state s'.

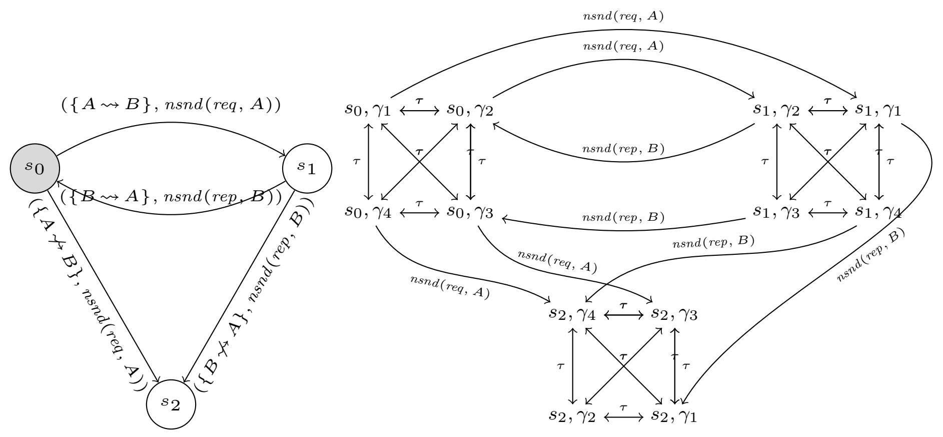
3.2 Unfolding a CLTS into an LTS

Mobility of nodes is modeled implicitly by a CLTS. For example, consider the CLTS in Fig. 1(a). In state s_0, A sends a req message; in case B is connected to A, there is a transition to state s_1, otherwise to state s_2. Thus, s_1 represents that a message has been received by B. In this state, B sends a rep message; in case A is connected to B, there is a transition to state s_0, otherwise to state s_2, which is a deadlock state. Hence, when a MANET has the state s, the underlying topology can change arbitrarily, but a behavior (i.e., transition), is possible, if the underlying topology belongs to the network constraint that the behavior under consideration is restricted to. Thus the CLTS models the behavior of the MANET compactly mobility is modeled implicitly.

The given CLTS is unfolded into a labeled transition system (LTS), whereby the network constrains are omitted and the underlying topology is made explicit in the state. Transformations of the topology are modeled by means of τ-transitions. The heart of the paper, which is the model checking algorithm for the temporal logic

CACTL presented in Section 6, will be performed entirely at the level of CLTSs. Still we present the unfolding of a CLTS into an LTS here, to clarify the meaning of CLTSs, and to illustrate how the unfolding grows as the number of nodes in the MANET increases.

Let $Loc = \{A, B\}$, so that the possible topologies are $\gamma_1 = \{A \mapsto \{B\}, B \mapsto \emptyset\}$, $\gamma_2 = \{A \mapsto \{B\}, B \mapsto \{A\}\}$, $\gamma_3 = \{A \mapsto \emptyset, B \mapsto \{A\}\}$, and $\gamma_4 = \{A \mapsto \emptyset, B \mapsto \emptyset\}$. The three states s_0, s_1, s_2 are paired with the four possible topologies $\gamma_1, \gamma_2, \gamma_3, \gamma_4$, leading to twelve states in total, as shown in Fig. 1(b). Formally speaking, the given CLTS $\langle S, \Lambda, \rightarrow, s_0 \rangle$, where $\Lambda \subseteq \mathbb{C}^v(Loc) \times Act_\tau$ and $\rightarrow \subseteq S \times \Lambda \times S$, is unfolded into the LTS $\langle S \times \Gamma, Act_\tau, \rightarrow, s_0 \times \Gamma \rangle$, where Γ is the set of possible topologies over Loc and $\rightarrow \subseteq S \times \Gamma \times Act_\tau \times S \times \Gamma$ such that $s, \gamma \xrightarrow{\eta} s', \gamma$ if $s \xrightarrow{\mathcal{C},\eta} s'$ with $\gamma \in \mathcal{C}$, and $s, \gamma \xrightarrow{\tau} s, \gamma'$ for all $\gamma' \in \Gamma \setminus \{\gamma\}$. As illustrated, the number of states and transitions grows exponentially as the number of nodes increases.



(a) A CLTS.   (b) The corresponding unfolded LTS.

Fig. 1 A CLTS and its unfolded LTS. The τ-transitions model mobility of nodes.

3.3 Multi-hop constraints and restricted mobility

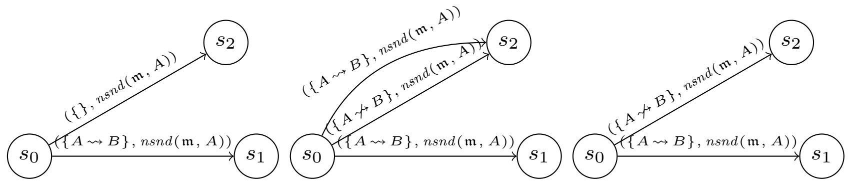
In MANETs two nodes can communicate if there is a multi-hop connection between them. Viewing a network topology as a directed graph, a multi-hop constraint is represented as a set of multihop connectivity pairs $\dashrightarrow: Loc \times Loc$. For instance, $A \dashrightarrow C$ denotes there exists a multi-hop connection from C to A, and consequently C can indirectly receive data from A. Let $\mathbb{M}(Loc)$ denote the set of multi-hop constraints that can be defined over network addresses in Loc, ranged over by $\mathcal{M}$. Each multi-hop constraint $\mathcal{M}$ represents the set of network topologies that satisfy multi-hop connectivity pairs in $\mathcal{M}$, i.e. $\gamma \in \mathcal{M}$ iff for each $\ell \dashrightarrow \ell'$ in $\mathcal{M}$ there is a multi-hop connection from ℓ' to ℓ in γ. Non-existence of a multi-path connection can be defined, however as usually behaviors depend on the existence of such connections rather than their non-existence, they are not considered in this paper.

As explained in Section 3.2, states in a CLTS do not hold information about the underlying topology. E.g., the transition $t_0 \xrightarrow{(\{A \rightsquigarrow B,\ A \not\rightsquigarrow C\}, \eta_1)} t_1$ implies that at the moment t_1 is reached, B is connected to A, and C is disconnected from A. In t_0 before this transition and in t_1 after this transition, the underlying topology can be anything. The arbitrary mobility implicitly modeled by the semantics can be restricted through a network constraint $\zeta \in \mathbb{C}$. As a result, the CLTS is restricted to transitions which conform to ζ, meaning that their network constraints do not include (dis)connectivity information that contradicts ζ. Conformance between network constraints $\mathcal{C}$ and ζ means that $\neg\mathcal{C} \cap \zeta = \emptyset$.

A multi-hop constraint $\mathcal{M}$ can restrict the arbitrary mobility in the semantics in two ways. First, it can restrict topology changes in states to those defined by $\mathcal{M}$. The behavioral model then only allows transitions $\xrightarrow{\mathcal{C},\eta}$ for which there exists a topology $\gamma \in \mathcal{C} \cap \mathcal{M}$. Thus, starting from a topology in $\mathcal{M}$ in the initial state, each state in any transition sequence has associated with it one or more topologies from $\mathcal{M}$. Second, it can require that the set of links implying $\mathcal{M}$ is the same in all associated topologies. A path $t_0(\mathcal{C}_1, \eta_1)t_1(\mathcal{C}_2, \eta_2)t_2 \ldots$ is said to be valid for a multi-hop constraint $\mathcal{M}$ if there exists a sequence $\gamma, \gamma_1, \gamma_2, \ldots \in \mathcal{M}$ such that $\gamma_i \in \mathcal{C}_i$ and γ is a subgraph of γ_i for all $i = 1, 2, \ldots$. Formally $\mathcal{C}^+_\Gamma(\gamma) \cap \neg\mathcal{C}_i = \emptyset$ means the links of γ, extracted by $\mathcal{C}^+_\Gamma$, are not disconnected in topologies of $\mathcal{C}_i$, and consequently there exists a $\gamma_i \in \mathcal{C}_i$ such that γ is a subgraph of γ_i. For instance, with the first approach the path $t_0 \xrightarrow{(\{A \rightsquigarrow B,\ A \not\rightsquigarrow C\}, \eta_1)} t_1 \xrightarrow{(\{B \not\rightsquigarrow A,\ B \not\rightsquigarrow C\}, \eta_2)} t_2$ is allowed for $\{A \dashrightarrow C\}$, since both $\{A \rightsquigarrow B,\ A \not\rightsquigarrow C\}$ and $\{B \not\rightsquigarrow A,\ B \not\rightsquigarrow C\}$ have a topology in common with $\{A \dashrightarrow C\}$. But it is not allowed with the second approach, since the constraint of the first transition forbids a direct connection from A to C, while the constraint of the second transition forbids an indirect connection from A to C via B. The second interpretation of topology restrictions works better for our verification purposes: a temporal property can be examined under the assumption that a stable multi-hop connection exists.

3.4 Reliable versus unreliable communication

Different behaviors for the communication primitives can be defined with the help of appropriate network constraints (see Fig. 2). Suppose that node A sends a message, and B is the only neighbor (waiting to receive). If B is connected to A and communication is reliable, then B will receive the message, as indicated in Fig. 2(c). If on the other hand communication is unreliable, then B may or may not receive the message, as modeled in Fig. 2(b). In this case we can infer the possible neighbors of A that did not receive the message. Such information can be hidden by merging information $A \rightsquigarrow B$ and $A \not\rightsquigarrow B$ and using the general network constraint $\{\}$, as in Fig. 2(a). By using appropriate operational rules, either of the three behaviors in Fig. 2 can be defined.


s_0
s_1
s_2
$(\{\}, nsnd(\mathfrak{m}, A))$
$(\{A \rightsquigarrow B\}, nsnd(\mathfrak{m}, A))$
s_0
s_1
s_2
$(\{A \rightsquigarrow B\}, nsnd(\mathfrak{m}, A))$
$(\{A \not\rightsquigarrow B\}, nsnd(\mathfrak{m}, A))$
$(\{A \rightsquigarrow B\}, nsnd(\mathfrak{m}, A))$
s_0
s_1
s_2
$(\{A \not\rightsquigarrow B\}, nsnd(\mathfrak{m}, A))$
$(\{A \rightsquigarrow B\}, nsnd(\mathfrak{m}, A))$

(a) Unreliable communication, no knowledge of neighbors (b) Unreliable communication (c) Reliable communication

Fig. 2 Modeling different communication behaviors: s_0 represents a state in which A broadcasts its data, s_1 a state after a successful transmission of data from A to B, and s_2 a state after an unsuccessful communication.

4 Reliable Restricted Broadcast Process Theory

The process algebra *RBPT* [17, 18] aims at the specification and verification of MANET protocols in a compositional way, with CLTSs as the underlying semantic model. In *RBPT* the communication primitive is defined as in Fig. 2(a). However, in the current setting it is more appealing to detect malfunctions of a MANET protocol caused by a conceptual mistake in the protocol design, rather than by unreliable communication. Therefore, we consider reliable communication here, and refer to the process algebra as *Reliable RBPT (RRBPT)*. The main difference between *RRBPT* and *RBPT* is in their operational rules: in *RRBPT*, nodes lose a communication only when they are disconnected. Furthermore, *RRBPT* introduces a novel operator to examine the status of a link, and upon its (non-)existence adapts its behavior accordingly to successfully accomplish its tasks. This operator abstracts away the neighbor discovery service implemented at the network layer, by which a node becomes aware of its connection topology discovery. The sensing operator obviates the explicit modeling of this service, and consequently makes modeling of protocols using such a service easier. We provide a brief overview of *RRBPT*, based on [17, 18].

4.1 Syntax

Let Msg denote a set of messages communicated over a network, ranged over by $\mathfrak{m}$. Furthermore, $\mathcal{A}$ denotes a countably infinite set of process names which are used as recursion variables in recursive specifications. Let $nsnd : Msg \times Loc$ and $nrcv : Msg$ represent *network send* and *network receive* actions and $snd, rcv : Msg$ *protocol send* and *protocol receive* actions. Let moreover $IAct$ be a set of internal actions, ranged over by i. The syntax of *RRBPT* is given by the following grammar, where ℓ ranges over Loc:

$$t ::= 0 \mid \alpha.t \mid t + t \mid [\![t]\!]_\ell \mid t \parallel t \mid A(d),\, A(d:D) \overset{def}{=} t \mid$$
$$sense(\ell, t, t) \mid (\nu\ell)t \mid \tau_m(t) \mid \partial_m(t)$$

The deadlock process is modeled by 0. The process $\alpha.t$ performs action α and then behaves as process t, where α is either an internal action or a protocol send/receive action $snd(\mathfrak{m})/rcv(\mathfrak{m})$. Internal actions are useful in modeling the interactions of a process with other applications running on the same node. Protocol send/receive actions specify the interaction of a process with its data-link layer protocols: these protocols are responsible for transferring messages reliably throughout the network. These actions are turned into their corresponding network ones via the semantics (see Section 4.2): the send action $nsnd(\mathfrak{m}, \ell)$ denotes that the message $\mathfrak{m}$ is transmitted from a node with the address ℓ, while the receive action $nrcv(\mathfrak{m})$ denotes that the message $\mathfrak{m}$ is ready to be received.

The process t_1+t_1 behaves non-deterministically as t_1 or t_2. The simplest form of a MANET is a node, represented by the deployment operator $[\![t]\!]_\ell$, denoting process t deployed on a node with the known network address $\ell \neq ?$ (where $?$ denotes the unknown address). A MANET can be composed by putting MANETs in parallel using $\|$; the nodes communicate with each other by reliable restricted broadcast. A process name is specified by $A \overset{def}{=} t$ where $A \in \mathcal{A}$ is a name. We restrict to so-called guarded process specifications: each occurrence of A in t must be within the scope of an action prefix.

As a running example, $P \overset{def}{=} init.snd(req).rcv(rep).succ.P$ denotes a process that recursively broadcasts a message req after performing the internal action $init$, waits to receive a rep and then performs an internal action $succ$; and $Q \overset{def}{=} rcv(req).snd(rep).Q$ a process that recursively receives a message req and then replies by sending rep. The internal action $init$ represents initialization of a route discovery from an upper layer application, and $succ$ a route discovery notification to an upper layer application. The network process $[\![P]\!]_A \parallel [\![Q]\!]_B$ specifies an ad hoc network composed of two nodes with network addresses A and B deploying processes P and Q, respectively.

MANET protocols may behave based on the (non-)existence of a link. A neighbor discovery service can be implemented at the network layer, by periodically sending *hello* messages and acknowledging such messages received from a neighbor. The sensing operator $sense(\ell', t_1, t_2)$ examines the status of the link from the node with address ℓ' to the node, say with address ℓ, that the sensing is executed on; in case of its existence it behaves as t_1, and otherwise as t_2. For instance, the term $[\![sense(\ell', t_1, t_2)]\!]_\ell$ examines the existence of the link $\ell' \rightsquigarrow \ell$, and then behaves accordingly. The hide operator $(\nu\ell)t$ conceals the address ℓ in the process t, by renaming this address to $?$ in network send/receive actions. For each message $m \in Msg$, the abstraction operator $\tau_m(t)$ renames network send/receive actions over messages of type m to τ, and the encapsulation operator $\partial_m(t)$ forbids receiving messages of type m. Let $\tau_{\{m_1,\ldots,m_n\}}(t)$ and $\partial_{\{m_1,\ldots,m_n\}}(t)$ denote $\tau_{m_1}(\ldots(\tau_{m_n}(t))\ldots)$ and $\partial_{m_1}(\ldots(\partial_{m_n}(t))\ldots)$.

For example, $\tau_{Msg}(\partial_{Msg}([\![P]\!]_A \parallel [\![Q]\!]_B))$ specifies an isolated MANET that cannot receive any message from the environment, while its communications (i.e. send actions) are abstracted away.

Terms should be grammatically well-defined, meaning that processes deployed at a network address are only defined by action prefix, choice, sense and process names.

4.2 Operational semantics

Given a MANET, the operational rules in Tables 1 and 2 induce a CLTS with transitions of the form $t \xrightarrow{\beta} t'$, where $\beta \in \mathbb{C}^v(Loc) \times (NAct \cup IAct \cup \{\tau\})$. Here $NAct$ denotes the set of network send and receive actions, and $IAct$ the set of internal actions. In these rules, $t \not\xrightarrow{(\mathcal{C},\ rcv(\mathfrak{m}))}$ denotes that there exists no t' such that $t \xrightarrow{(\mathcal{C},\ rcv(\mathfrak{m}))} t'$ for some $\mathcal{C}$.

Table 1 Semantics of prefix, choice, process name and sensing operators.

$$\frac{}{\alpha.t \xrightarrow{(\{\},\alpha)} t} : Prefix \qquad \frac{t_1 \xrightarrow{(\mathcal{C},\alpha)} t_1'}{sense(\ell,t_1,t_2) \xrightarrow{(\{\ell \rightsquigarrow ?\}\cup\mathcal{C},\alpha)} t_1'} : Sen_1$$

$$\frac{t_i \xrightarrow{(\mathcal{C},\alpha)} t_i' \;\; i \in \{1,2\}}{t_1 + t_2 \xrightarrow{(\mathcal{C},\alpha)} t_i'} : Choice \qquad \frac{t_2 \xrightarrow{(\mathcal{C},\alpha)} t_2'}{sense(\ell,t_1,t_2) \xrightarrow{(\{\ell \not\rightsquigarrow ?\}\cup\mathcal{C},\alpha)} t_2'} : Sen_2$$

$$\frac{t \xrightarrow{(\mathcal{C},\alpha)} t' \quad A \overset{def}{=} t}{A \xrightarrow{(\mathcal{C},\alpha)} t'} : Inv \qquad \frac{t_1 \not\xrightarrow{(\mathcal{C},\ rcv(\mathfrak{m}))}}{sense(\ell,t_1,t_2) \xrightarrow{(\{\ell \rightsquigarrow ?\},\ rcv(\mathfrak{m}))} t_1} : Sen_3$$

$$\frac{t_2 \not\xrightarrow{(\mathcal{C},\ rcv(\mathfrak{m}))}}{sense(\ell,t_1,t_2) \xrightarrow{(\{\ell \not\rightsquigarrow ?\},\ rcv(\mathfrak{m}))} t_2} : Sen_4$$

Prefix indicates execution of a prefix action (which is either a protocol or an internal action). *Choice* specifies the non-deterministic behavior of the choice operator in terms of its operands. *Inv* defines the behavior of a process name in terms of the right-hand side of its definition $A \overset{def}{=} t$. *Int* defines when a node performs an internal action. Rules Sen_{1-4} explain the behavior of sense operator; in case there is a link from the node with address ℓ to the node that running the sense operator, currently its address is unknown, it behaves like t_1, otherwise t_2. Therefore, the link status is combined with the network constraint $\mathcal{C}$ generated by its first or second term argument, as given by $Sen_{1,2}$ respectively. Rules $Sen_{3,4}$ define that if a process does not have any enabled receive action $rcv(\mathfrak{m})$, then receiving the message has no effect on the node behavior. Again, the network constraints that reflect the link status are generated by $Sen_{3,4}$. These rules together with Rcv_3 make the nodes *input enabled*, meaning that a node not ready to receive a message will drop it.

For instance, $sense(B, rcv(req).0, snd(req).0)$ generates three transitions: by *Prefix* and Sen_1, generates a $(\{B \rightsquigarrow ?\},\ rcv(req))$-transition, by *Prefix* and Sen_2, a $(\{B \not\rightsquigarrow ?\},\ snd(req))$-transition, and by Sen_3, a $(\{B \not\rightsquigarrow ?\},\ rcv(req))$-transition.

The rule *Int* indicates that a node progresses when the deployed process on the node performs an internal action. Interaction between the process t and its datalink layer is specified by the rule *Snd*: when t broadcasts a message, it is delivered to the nodes in its transmission range disregarding their readiness. Rcv_1 spec-

Table 2 Semantics of deployment, parallel, abstraction, encapsulation, and hide operators.

$$\frac{t \xrightarrow{(\mathcal{C},\, i)} t'}{[\![t]\!]_\ell \xrightarrow{(\mathcal{C},\, i)} [\![t']\!]_\ell} : \mathit{Int},\ \ i \in \mathit{IAct}$$

$$\frac{t \xrightarrow{(\mathcal{C},\, snd(\mathfrak{m}))} t'}{[\![t]\!]_\ell \xrightarrow{(\mathcal{C}[\ell/?],\, nsnd(\mathfrak{m},\ell))} [\![t']\!]_\ell} : \mathit{Snd} \qquad \frac{t \xrightarrow{(\mathcal{C},\eta)} t' \quad \eta \neq nrcv(m)}{\partial_m(t) \xrightarrow{(\mathcal{C},\eta)} \partial_m(t')} : \mathit{Encap}$$

$$\frac{t \xrightarrow{(\mathcal{C},\, rcv(\mathfrak{m}))} t'}{[\![t]\!]_\ell \xrightarrow{(\mathcal{C}[\ell/?]\cup\{?\rightsquigarrow\ell\},\, nrcv(\mathfrak{m}))} [\![t']\!]_\ell} : \mathit{Rcv}_1 \qquad \frac{t \xrightarrow{\beta} t'}{(\nu\ell)t \xrightarrow{\beta[?/\ell]} (\nu\ell)t'} : \mathit{Hid}$$

$$\frac{t \xrightarrow{(\mathcal{C},\, rcv(\mathfrak{m}))} t'}{[\![t]\!]_\ell \xrightarrow{(\mathcal{C}[\ell/?]\cup\{?\not\rightsquigarrow\ell\},\, nrcv(\mathfrak{m}))} [\![t]\!]_\ell} : \mathit{Rcv}_2 \qquad \frac{t \not\xrightarrow{(\mathcal{C},\, rcv(\mathfrak{m}))}}{[\![t]\!]_\ell \xrightarrow{(\{\},\, nrcv(\mathfrak{m}))} [\![t]\!]_\ell} : \mathit{Rcv}_3$$

$$\frac{t_i \xrightarrow{(\mathcal{C},\eta)} t'_i \quad i \in \{1,2\},\ \eta \in \mathit{IAct} \cup \{\tau\}}{t_1 \parallel t_2 \xrightarrow{(\mathcal{C},\eta)} t'_i \parallel t_{3-i}} : \mathit{Par} \qquad \frac{t \xrightarrow{(\mathcal{C},\eta)} t'}{\tau_m(t) \xrightarrow{(\mathcal{C},\tau_m(\eta))} \tau_m(t')} : \mathit{Abs}$$

$$\frac{t_1 \xrightarrow{(\mathcal{C}_1,\, nrcv(\mathfrak{m}))} t'_1 \quad t_2 \xrightarrow{(\mathcal{C}_2,\, nrcv(\mathfrak{m}))} t'_2}{t_1 \parallel t_2 \xrightarrow{(\mathcal{C}_1\cup\mathcal{C}_2,\, nrcv(\mathfrak{m}))} t'_1 \parallel t'_2} : \mathit{Sync}_1 \qquad \frac{t \xrightarrow{(\mathcal{C},\eta)} t'}{t \xrightarrow{(\mathcal{C}',\eta)} t'} : \mathit{Exe},\ \ \mathcal{C} \subseteq \mathcal{C}'$$

$$\frac{t_i \xrightarrow{(\mathcal{C}_1,\, nsnd(\mathfrak{m},\ell))} t'_i \quad t_{3-i} \xrightarrow{(\mathcal{C}_2,\, nrcv(\mathfrak{m}))} t'_{3-i} \quad i \in \{1,2\}}{t_1 \parallel t_2 \xrightarrow{(\mathcal{C}_1\cup\mathcal{C}_2[\ell/?],\, nsnd(\mathfrak{m},\ell))} t'_1 \parallel t'_2} : \mathit{Sync}_2$$

ifies that a process t with an enabled receive action can perform it successfully, if it has a link to a sender (not currently known). In contrast, an enabled receive action cannot be performed, if the node is disconnected from the sender (not currently known). However, to let an enabled receive action be unaffected by out-of-range activities, the node still performs its receive action but its state is unchanged, as explained in Rcv_2. In rules $\mathit{Rcv}_{1,2}$, the network constraint $\mathcal{C}$ may have unknown addresses due to the sensing operators, which are replaced by the address of development operator, i.e., $\mathcal{C}[\ell/?]$. Therefore, by the application of Rcv_1, Sen_1, and Prefix, $[\![sense(B, rcv(req).0, snd(req).0)]\!]_A \xrightarrow{(\{B\rightsquigarrow A,\ ?\rightsquigarrow A\},\ nrcv(req))} 0$ results. If a node does not have any enabled receive action $rcv(\mathfrak{m})$, then receiving the message has no effect on the node behavior, as explained by Rcv_3. Rule Snd together with $\mathit{Rcv}_{2,3}$ and $\mathit{Sen}_{3,4}$ define local-broadcast as *non-blocking*, meaning that no sender can be delayed in transmitting a message because of a disconnected or non-ready receiver.

Rule Sync_1 synchronizes the receive actions of processes t_1 and t_2 on message $\mathfrak{m}$, while combining together their (dis)connectivity information in network constraints $\mathcal{C}_1$ and $\mathcal{C}_2$. Rule Sync_2 specifies how a communication occurs between a receiving and a sending process. This rule combines the network constraints, while the unknown location (in the network constraint of the receiving process) is instantiated with the concrete address of the sender. In $Sync_{1,2}$ it is required that the union of network constraints on the transition in the conclusion be well-formed. By Par, a process evolves when a subprocess evolves by performing an internal or silent action. Exe explains that a behavior that is possible for a network constraint, is also possible for a more restrictive network constraint.

For instance, the MANET $\llbracket sense(B, rcv(req).0, snd(req).0)\rrbracket_A \parallel \llbracket snd(req).0\rrbracket_B$ can generate the $(\{B \rightsquigarrow A\},\ nsnd(req, B))$ transition induced by the deduction tree below, where $x \equiv sense(B, rcv(req).0, snd(req).0)$ and $y \equiv snd(req).0$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{rcv(req).0 \xrightarrow{(\{\},\ rcv(req))} 0}\ {:}Prefix}{x \xrightarrow{(\{B\rightsquigarrow ?\},\ rcv(req))} 0}\ {:}Sen_1}{\llbracket x\rrbracket_A \xrightarrow{(\{B\rightsquigarrow A,\ ?\rightsquigarrow A\},\ nrcv(req))} \llbracket 0\rrbracket_A}\ {:}Rcv_1 \qquad \dfrac{\dfrac{}{y \xrightarrow{(\{\},\ snd(req))} 0}\ {:}Prefix}{\llbracket y\rrbracket_B \xrightarrow{(\{\},\ nsnd(req),B)} \llbracket 0\rrbracket_B}\ {:}Snd}{\llbracket x\rrbracket_A \parallel \llbracket y\rrbracket_B \xrightarrow{(\{B\rightsquigarrow A\},\ nsnd(req,B))} \llbracket 0\rrbracket_A \parallel \llbracket 0\rrbracket_B}\ {:}Sync_2$$

Note that by $Prefix$, Sen_1, and Rcv_2, $\llbracket x\rrbracket_A \xrightarrow{(\{B\rightsquigarrow A,\ ?\not\rightsquigarrow A\}, nrcv(req))} \llbracket 0\rrbracket_A$. However this transition cannot participate with $\llbracket y\rrbracket_B \xrightarrow{(\{\},\ nsnd(req,A))} \llbracket 0\rrbracket_B$ in rule $Sync_2$, since $\{B \rightsquigarrow A,\ ? \not\rightsquigarrow A\}[B/?] \cup \{\}$ is not well-formed. Therefore, the sensing operator enforces the node A to communicate with B in case A is connected to B.

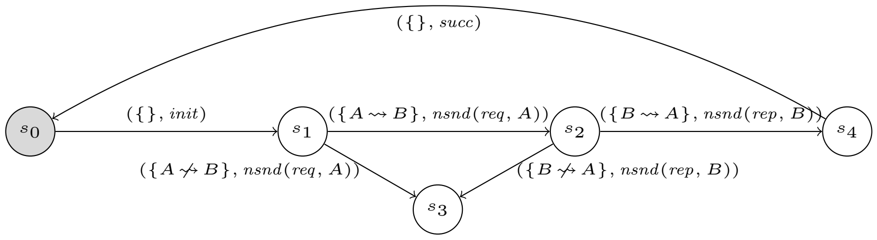
Rule Hid replaces every occurrence of ℓ in the network constraint and action of β by ?, and hence hides activities of a node with address ℓ from external observers. According to Abs, the abstraction operator τ_m converts all network send and receive actions with a message of type m to τ and leaves other actions unaffected, as defined by the function $\tau_m(\eta)$. The encapsulation operator ∂_m disallows all network receive actions on messages of type m, as specified by $Encap$.

The operational semantics rules of receive actions and parallel composition have been modified compared to the lossy framework [17, 18]: Rcv_{1-3} specify the locality of a receiver node with respect to the sender (connected, disconnected, unknown). Par prevents evolution of sub-networks on network actions and enforces all nodes to specify their localities with respect to the sender before evolving the whole network via $Sync_{1,2}$.

The derived CLTS of our running example $\partial_{Msg}(\llbracket P\rrbracket_A \parallel \llbracket Q\rrbracket_B)$ is given in Fig. 3.

5 Constrained Action Computation Tree Logic

As the behavior of a MANET depends on the topology of the underlying network, the properties of a MANET protocol may depend on constraints on this topology. In verifying whether a temporal logic formula holds for a certain MANET, we may only want to explore paths that satisfy certain connectivity conditions with regard to the underlying topology, such as a direct link between two nodes, or the existence of a multi-hop connection between two nodes. CLTSs provide a suitable platform to verify topology-dependent properties, using the (dis)connectivity information encoded



$(\{\}, succ)$
$(\{\}, init)$
$(\{A \rightsquigarrow B\}, nsnd(req, A))$
$(\{B \rightsquigarrow A\}, nsnd(rep, B))$
$(\{A \not\rightsquigarrow B\}, nsnd(req, A))$
$(\{B \not\rightsquigarrow A\}, nsnd(rep, B))$
s_0
s_1
s_2
s_3
s_4

Fig. 3 The CLTS associated to $\partial_{Msg}(\llbracket P \rrbracket_A \parallel \llbracket Q \rrbracket_B)$.

in transition labels. Transitions are traversed to investigate a behavioral property as long as (dis)connectivity information implies topology requirements on which the property depends. To this aim single- and multi-hop constraints are used, taking mobility into account in different ways. Single-hop constraints limit the one-step movements of nodes and hence, restrict transitions that can be traversed, while multi-hop constraints limit the mobility scenarios of nodes and thus, restrict the paths that need to be investigated (see Section 3.3). Since a CLTS is unfolded to an LTS in which mobility is modeled explicitly by pairing a state with its underlying topology (see Section 3.2), a behavioral property can be decorated with connectivity conditions while the topology is captured by atomic propositions on states (cf. [13]).

Action Computational Tree Logic (ACTL) [11] parameterizes the temporal operators *next* $\mathbf{X}$ and *until* $\mathbf{U}$ from CTL [7] with a set of actions. It provides a general framework for verifying properties in process algebra [10]. ACTLW [25] further enriches the *until* operator from ACTL and adds an (enriched) *unless* operator $\mathbf{W}$. We note that in the action-based logic settings, in contrast to CTL, $\mathbf{EW}$ cannot readily be defined in terms of $\mathbf{AU}$. In Section 5.2 we will define a temporal logic for MANETs in which the *until* and *unless* operators from ACTLW are preceded by either the path quantifier *Exists* $\mathbf{E}$ or *All* $\mathbf{A}$ from ACTLW decorated with multi-hop constraints.

5.1 Motivating example

Consider the CLTS of $\partial_{Msg}(\llbracket P \rrbracket_A \parallel \llbracket Q \rrbracket_B)$ in Fig. 3. The internal action $init$ denotes a request from an application at node A that initializes a route discovery process to node B, and $succ$ denotes the successful termination of that route discovery process. To verify correctness of the route discovery protocol, taking into account mobility of nodes, any route discovery initialization must terminate successfully whenever there is a multi-hop connection from A and B as well as from B to A. This property can be specified by: *for all mobility scenarios in which $A \dashrightarrow B$ and $B \dashrightarrow A$ are valid for a sufficiently long period of time, each occurrence of $init$ is eventually followed by an occurrence of $succ$*. This property is satisfied by the CLTS in Fig. 3. In particular, consider the following two paths, where s_3 is a deadlock:

$$s_0(\{\}, \mathit{init})s_1(\{A \not\rightsquigarrow B\}, \mathit{nsnd}(\mathit{req}, A))s_3$$

$$s_0(\{\}, \mathit{init})s_1(\{A \rightsquigarrow B\}, \mathit{nsnd}(\mathit{req}, A))s_2(\{B \not\rightsquigarrow A\}, \mathit{nsnd}(\mathit{rep}, B))s_3.$$

Although in both paths init is not followed by an occurrence of succ, these paths do not violate the property above. Namely, to satisfy $A \dashrightarrow B$ and $B \dashrightarrow A$ in a network of two nodes, A and B should be directly connected to each other. However, these links get disconnected by the network constraint $\{A \not\rightsquigarrow B\}$ in the first and $\{B \not\rightsquigarrow A\}$ in the second path. Therefore, these paths are not valid for the multi-hop constraint $\{A \dashrightarrow B, B \dashrightarrow A\}$.

Suppose a regulator node $[\![R]\!]_C$ with

$$R \overset{def}{=} \mathit{rcv}(\mathit{req}).\mathit{snd}(\mathit{req}).R + \mathit{rcv}(\mathit{rep}).\mathit{snd}(\mathit{rep}).R$$

is added. The resulting network $\partial_{Msg}([\![P]\!]_A \parallel [\![Q]\!]_B \parallel [\![R]\!]_C)$ still satisfies the above property. However, if R is replaced by the erroneous protocol

$$E \overset{def}{=} \mathit{rcv}(\mathit{req}).\mathit{snd}(\mathit{req}).E + \mathit{rcv}(\mathit{req}).E + \mathit{rcv}(\mathit{rep}).\mathit{snd}(\mathit{rep}).E + \mathit{rcv}(\mathit{rep}).E$$

that may drop a packet it receives, then the new MANET does not satisfy the property.[1] For example, the path which ends in the deadlock state $\partial_{Msg}([\![\mathit{rcv}(\mathit{rep}).P]\!]_A \parallel [\![Q]\!]_B \parallel [\![E]\!]_C$

$$\begin{aligned}&\partial_{Msg}([\![P]\!]_A \parallel [\![Q]\!]_B \parallel [\![E]\!]_C)\\ &\qquad(\{\}, \mathit{init})\partial_{Msg}([\![\mathit{snd}(\mathit{req}).\mathit{rcv}(\mathit{rep}).P]\!]_A \parallel [\![Q]\!]_B \parallel [\![E]\!]_C)\\ &\qquad(\{A \rightsquigarrow C,\ A \not\rightsquigarrow B\}, \mathit{nsnd}(\mathit{req}, A))\partial_{Msg}([\![\mathit{rcv}(\mathit{rep}).P]\!]_A \parallel [\![Q]\!]_B \parallel [\![E]\!]_C)\end{aligned}$$

satisfies the multi-hop constraint $\{A \dashrightarrow B,\ B \dashrightarrow A\}$ in a network of three nodes, namely A, B and C, but the occurrence of init is not followed by an occurrence of succ. For instance, with regard to the topology $\gamma = \{A \mapsto \{C\}, B \mapsto \{C\}, C \mapsto \{A, B\}\}$ in $\{A \dashrightarrow B,\ B \dashrightarrow A\}$ (note that the network constraints $\{\}$ and $\{A \rightsquigarrow C,\ A \not\rightsquigarrow B\}$ agree with γ), C may drop the req packet.

5.2 CACTL syntax

The temporal logic *Constrained Action Computation Tree Logic* (CACTL) allows to express topology-dependent properties of MANET protocols. The path quantifier *All* is parameterized by a multi-hop constraint over the topology, which specifies the pre-condition required for paths of a state to be inspected. A typical example of such a constraint is the existence of a (topological) path between two nodes. A pre-condition restricts mobility scenarios, implicitly modeled in the semantics, to ones for which the pre-condition holds. That is, paths for which the pre-condition does not holds, are not required to satisfy the path formula expressed by the state formula under consideration. However, the path quantifier *Exists* is defined as before and inspects all paths of a state to find at least one satisfying the specified (path) property. By combining *All* and *Exists* path quantifiers, the decorated version of *Exists*, pre-conditioned by a multi-hop constraint, can be specified to also include states that have no path for

[1] With the semantics of the logic CACTL as given in [16], the property is satisfied, so that the error caused by E is not detected.

which the pre-condition holds. In other words, if a state has no path for which the pre-condition holds, then it satisfies the formula spontaneously.

Moreover, to restrict mobility scenarios in a more concrete way, the satisfaction relation is parameterized with single-hop constraints. This parameter expresses the (non-)existence of communication links; nodes can only move in such a way that the specified links do not change. This is achieved by only traversing transitions that conform to the specified links.

A CACTL formula may include constants *true* and *false*, actions $\eta \in Act_\tau$, standard Boolean operators $\neg$, $\wedge$ and $\vee$, path quantifiers $\mathbf{A}^{\mu}$ parameterized by constraints over topologies and $\mathbf{E}$, and temporal operators $\mathbf{U}$ and $\mathbf{W}$.

Let $\eta \in Act_\tau$, and $\ell, \ell' \in Loc$. *Action formula* χ, *topology formula* μ, *state formula* ϕ (also called *CACTL formula*), and *path formula* ψ are defined by the grammars:

$$\begin{aligned}
\chi &::= \ true \mid \eta \mid \neg\chi \mid \chi \wedge \chi' \\
\mu &::= \ true \mid \ell \dashrightarrow \ell' \mid \mu \wedge \mu' \\
\phi &::= \ true \mid \neg\phi \mid \phi \wedge \phi' \mid \mathbf{E}\psi \mid \mathbf{A}^{\mu}\psi \\
\psi &::= \ \phi\,{}_{\chi}\mathbf{U}_{\chi'}\,\phi' \mid \phi\,{}_{\chi}\mathbf{W}_{\chi'}\,\phi'
\end{aligned}$$

Each temporal operator should be preceded by a path quantifier, forming a *CACTL operator*. Action and path formulae are the same as in ACTLW [25]. Topology formulae to restrict state formulae and require multi-hop connections are novel. Each topology formula induces the multi-hop constraint made of multi-hop connections participating in the topology formula under consideration. In the reminder of this paper, we use the notions of topology formula and multi-hop constraint interchangeably. For the topology formula μ and topology γ, assume $\gamma \in \mu$ denotes $\gamma \in \mathcal{M}$, where $\mathcal{M}$ is the multi-hop constraint induced by μ.

Let $T \equiv \langle S, A, \rightarrow, s_0\rangle$ be a CLTS. A *path* π in T from a state $t_0 \in S$ is a sequence of transitions $t_0(\mathcal{C}_1,\eta_1)t_1(\mathcal{C}_2,\eta_2)t_2\ldots$ where $\forall i \geq 1\,((t_{i-1},(\mathcal{C}_i,\eta_i),t_i) \in \rightarrow)$. A path is said to be *maximal* if it either is infinite or ends in a deadlock state, meaning that there are no outgoing transitions. Furthermore, a path is called ζ*-path* if $\mathcal{C}_i$ conforms to ζ, i.e., $\forall i \geq 1(\neg\mathcal{C}_i \cap \zeta = \emptyset)$. State and path formulae are interpreted with regard to a network constraint $\zeta \in \mathbb{C}^v(Loc)$, meaning that only maximal ζ-paths are considered in the evaluation of such formulae. A network constraint $\mathcal{C}' \in \mathbb{C}(Loc)$ is said to be invalid for (or violate) μ, if $\forall\gamma \in \mu(\mathcal{C}^+_T(\gamma) \cap \neg\mathcal{C}' \neq \emptyset)$. We say a path violates μ in state t_i, if the accumulated network constraints before reaching to t_i violates μ, i.e., $\forall\gamma \in \mu(\mathcal{C}^+_T(\gamma) \cap \neg\bigcup_{1\leq j\leq i}\mathcal{C}_j \neq \emptyset)$. A path is said to be invalid for μ (or violate μ), if it is invalid in some state over the path.

The path quantifier $\mathbf{E}$ requires that the given path formula is satisfied for at least one maximal ζ-path starting in the given state, while $\mathbf{A}^{\mu}$ requires the property holds for all ζ-paths that are valid for the multi-hop constraint induced by the topology formula μ.

A state t that satisfies a CACTL formula ϕ is called a ϕ-state, and a transition with an action from χ is called a χ-transition. A χ-transition that ends in a ϕ-state is called a (χ,ϕ)-transition. A path with an initial ϕ-state and only (χ,ϕ)-transitions is called a (χ,ϕ)-path. The until operator $\phi\,{}_{\chi}\mathbf{U}_{\chi'}\,\phi'$ is satisfied by maximal ζ-paths that start at an initial ϕ-state and perform a finite sequence of (χ, ϕ)-transitions, until a (χ', ϕ')-transition is performed. The unless (or weak until) operator $\phi\,{}_{\chi}\mathbf{W}_{\chi'}\,\phi'$ extends $\phi\,{}_{\chi}\mathbf{U}_{\chi'}\,\phi'$ by moreover allowing infinite ζ-paths which are a (χ, ϕ)-path. We note that, in contrast to CTL, $\mathbf{EW}$ cannot readily be defined in terms of $\mathbf{AU}$, because the states satisfying ϕ and ϕ' must be visited by transitions carrying actions from χ and χ' respectively.

For example, the state formula

$$\mathbf{A}^{\,true}(true\,{}_{\neg init}\mathbf{W}_{init}\,\mathbf{A}^{\,A\dashrightarrow B\wedge B\dashrightarrow A}(true\,{}_{\tau}\mathbf{U}_{succ}\,true))$$

indicates that an action *init* is always eventually followed by an action *succ*, possibly after some communications (specified by τ), under the condition that from the moment that *init* occurs on there exists a multi-hop connection from A to B and from B to A. This formula is satisfied by the initial state of the CLTS given in Fig. 3 after all communications have been turned into τ with the help of abstraction operator, i.e., by $\tau_{Msg}(\partial_{Msg}([\![P]\!]_A \parallel [\![Q]\!]_B))$.

By combining $\mathbf{E}$ and $\mathbf{A}^{\mu}$, $\mathbf{E}^{\mu}$ can be derived as $\mathbf{E}^{\mu}\varphi = \mathbf{E}\varphi \vee \mathbf{A}^{\,\mu}\varphi$. Intuitively, $\mathbf{E}^{\mu}\varphi$ is satisfied by a state which either has at least one maximal ζ-path that it satisfies the path formula φ, or has maximal ζ-paths that all violate μ. Consequently, $\mathbf{EF}^{\mu}_{\chi}\phi$ is satisfied by a state that either has at least one maximal ζ-path which contains a (χ, ϕ)-transition or all its maximal ζ-paths are invalid for μ.

Other CACTL formulae can be derived from $\mathbf{E}$, $\mathbf{A}$, $\mathbf{U}$ and $\mathbf{W}$ following the approach of [25]:

$$\begin{array}{ll}
\mathbf{EX}_{\chi}\phi = \mathbf{E}(true\,{}_{false}\mathbf{U}_{\chi}\,\phi) & \mathbf{AX}_{\chi}\phi = \mathbf{A}^{\,true}(true\,{}_{false}\mathbf{U}_{\chi}\,\phi) \\
\mathbf{EF}_{\chi}\phi\; = \mathbf{E}(true\,{}_{true}\mathbf{U}_{\chi}\,\phi) & \mathbf{AF}^{\mu}_{\chi}\phi\; = \mathbf{A}^{\,\mu}(true\,{}_{true}\mathbf{U}_{\chi}\,\phi) \\
\mathbf{EG}_{\chi}\phi = \mathbf{E}(\phi\,{}_{\chi}\mathbf{W}_{false}\,false) & \mathbf{AG}^{\mu}_{\chi}\phi = \mathbf{A}^{\,\mu}true(\phi\,{}_{\chi}\mathbf{W}_{false}\,false)
\end{array}$$

Intuitively, $\mathbf{AX}_{\chi}\phi$ is satisfied by states of which all maximal ζ-paths start with a (χ, ϕ)-transition, $\mathbf{AF}^{\mu}_{\chi}\phi$ by states of which all maximal ζ-paths either contain a (χ, ϕ)-transition or in some state violate μ, and $\mathbf{AG}^{\mu}_{\chi}\phi$ by states of which all maximal ζ-paths visit only ϕ-states and perform only χ-transitions as long as μ is valid.

5.3 CACTL semantics

Let $T \equiv \langle S, \Lambda, \rightarrow, s_0\rangle$ be a CLTS. For a path π of T, assume π^s_i, $\pi^{\mathcal{C}}_i$ and π^{η}_i denote the ith state, network constraint and action on path π, while $len(\pi)$ denotes the number of transitions in π (which is ∞ if π is infinite). So a finite path π is of the form $\pi^s_0(\pi^{\mathcal{C}}_1, \pi^{\eta}_1)\pi^s_1 \cdots (\pi^{\mathcal{C}}_{len(\pi)}, \pi^{\eta}_{len(\pi)})\pi^s_{len(\pi)}$. Let $\mathcal{C}^+_{\Gamma}(\gamma)$ extract all connectivity relations in topology γ.

Satisfaction by $\eta \in Act_{\tau}$ of action formula χ (written $\eta \models \chi$), by state t of state formula ϕ under network constraint ζ (written $t \models_{\zeta} \phi$), and by maximal path π of path formula ψ under network constraint ζ (written $\pi \models_{\zeta} \psi$), is defined inductively below.

$\eta \models \;\; true$ always

$$\begin{array}{ll}
\eta \models\ \eta' & \text{iff } \eta = \eta' \\
\eta \models\ \neg\chi & \text{iff } \eta \not\models \chi \\
\eta \models\ \chi \wedge \chi' & \text{iff } \eta \models \chi \wedge \eta \models \chi' \\
t \models_\zeta \mathit{true} & \text{always} \\
t \models_\zeta \neg\phi & \text{iff } t \not\models_\zeta \phi \\
t \models_\zeta \phi \wedge \phi' & \text{iff } t \models_\zeta \phi \wedge t \models_\zeta \phi' \\
t \models_\zeta \mathbf{E}\psi & \text{iff there exists a maximal } \zeta\text{-path } \pi \text{ such that } t = \pi^s_0 \wedge \pi \models_\zeta \psi \\
t \models_\zeta \mathbf{A}^{\mu}\psi & \text{iff for all maximal } \zeta\text{-paths } \pi \text{ with } t = \pi^s_0 \text{ either } \pi \models_\zeta \psi, \\
 & \text{or } \psi \equiv \phi\,_\chi\mathbf{V}_{\chi'}\,\phi'\text{, where } \mathbf{V} \in \{\mathbf{U}, \mathbf{W}\}\text{, and } \pi^s_0 \models_\zeta \phi \text{ and } \exists 1 \le j \le \mathit{len}(\pi)\ (\\
 & \forall\, 1 \le i \le j\, (\pi^s_i \models_\zeta \phi \ \wedge\ \pi^\eta_i \models \chi)\ \wedge\ \forall\gamma \in \mu\, (\mathcal{C}^+_\Gamma(\gamma) \cap \bigcup^j_{k=1} \neg\pi^{\mathcal{C}}_k \neq \emptyset)) \\
\pi \models_\zeta \phi\,_\chi\mathbf{U}_{\chi'}\,\phi' & \text{iff } \pi^s_0 \models_\zeta \phi \text{ and } \exists 1 \le j \le \mathit{len}(\pi)\ (\\
 & \forall\, 1 \le i < j\, (\pi^s_i \models_\zeta \phi \ \wedge\ \pi^\eta_i \models \chi)\ \wedge\ (\pi^s_j \models_\zeta \phi' \ \wedge\ \pi^\eta_j \models \chi')) \\
\pi \models_\zeta \phi\,_\chi\mathbf{W}_{\chi'}\,\phi' & \text{iff either } \pi \models_\zeta \phi\,_\chi\mathbf{U}_{\chi'}\,\phi', \\
 & \text{or } \pi^s_0 \models_\zeta \phi \text{ and } \forall 1 \le i \le \mathit{len}(\pi)\, (\pi^s_i \models_\zeta \phi \ \wedge\ \pi^\eta_i \models \chi)
\end{array}$$

The semantics of the until operator in ACTL, ACTLW and CACTL is somewhat different from CTL: $\mathbf{E}(\phi_\chi \mathbf{U}_{\chi'} \phi')$ in ACTLW requires to perform at least one action from χ' to reach a state satisfying ϕ', while $\mathbf{E}(\phi \mathbf{U} \phi')$ in CTL is also satisfied if the first state already satisfies ϕ'. Similar to ACTL but in contrast to ACTLW, our semantics of the until operator explicitly distinguishes silent from visible actions.

In [16] it is implicitly assumed that all nodes are initially disconnected and they move as little as possible to satisfy network constraints over transitions. Network constraints are accumulated over a path such that the last status of a link overwrites the previous one. A path is invalid for a multi-hop constraint if the accumulated network constraints do not imply it. This means that more paths are considered invalid for a topology formula, which reduces the distinguishing power of the until operator. For instance, the finite path $t_0 \xrightarrow{(\{A \rightsquigarrow C,\ A \not\rightsquigarrow B\}, \eta)} t_1$, where t_1 is a deadlock, is considered invalid for $A \dashrightarrow B$, since $\{A \rightsquigarrow C,\ A \not\rightsquigarrow B\}$ does not imply $A \dashrightarrow B$. However, for a topology where B is connected to C and C is connected to A, $A \dashrightarrow B$ as well as the network constraint on the transition is satisfied. Therefore, conceptual errors that are only visible over such invalid paths (see the example in Section 5.1) can not be caught. In this paper, we have boosted the distinguishing power of CACTL by adopting the new definition of invalid μ-path. Furthermore, in [16] $\mathbf{E}$-formulae are, similar to our $\mathbf{A}$-formulae, parameterized by a network constraint μ, so that such a formula is satisfied by a path that violates μ in some state, irrespective to the formula under consideration. Therefore a formula $\mathbf{E}(\phi_\chi \mathbf{U}_{\chi'} \phi')$ may be satisfied by a state with a maximal ζ-path that violates μ, while other maximal ζ-paths are valid for μ with no occurrence of a (χ', ϕ')-transition. Finally, in [16] unobservable actions (over all possible topologies, i.e., $(\{\}, \tau)$) are passed by while looking for (χ, ϕ)-transitions, similar to ACTL. This is essential for defining a branching bisimulation semantics with the same identification power as CACTL, as is done in [16]. However, the next operator cannot be derived from the until operator. Therefore here, similar to ACTLW, in the semantics of the until operator unobservable actions are treated in the same way as visible actions.

6 CACTL Model Checking Algorithm

We adapt the CTL model checking algorithm (see [9,8]) to CACTL. Model checking a CACTL formula φ on a CLTS under $\zeta \in \mathbb{C}$ starts with the smallest subformulae and works outwards toward φ. The procedure $\mathit{ModelCheck}(\mathit{CLTS}, \varphi, \zeta)$ returns those states in CLTS that satisfy φ under ζ. The pseudocode of the model checking algorithm's backbone, where each syntactic form of CACTL's grammar is provided with a dedicated subcall, is given in Fig. 4. In following Section, we explain about the details of each subcall.

```
**Procedure** $\mathit{ModelCheck}(\mathit{CLTS}, \varphi, \zeta)$
**Output**: The states in $\mathit{CLTS}$ that satisfy $\varphi$ under $\zeta$
**switch** $\varphi$ **do**
    **case** $\phi \wedge \phi'$
        $T_1 := \mathit{ModelCheck}(\mathit{CLTS}, \phi, \zeta)$;
        $T_2 := \mathit{ModelCheck}(\mathit{CLTS}, \phi', \zeta)$;
        **return** $T_1 \cap T_2$;
    **case** $\mathbf{E}(\phi\,_{\chi}\mathbf{U}_{\chi'}\,\phi')$
        **return** $\mathit{CheckEU}(\mathit{CLTS}, \chi, \phi, \chi', \phi', \zeta)$
    **case** $\mathbf{A}^{\mu}(\phi\,_{\chi}\mathbf{U}_{\chi'}\,\phi')$
        **return** $\mathit{CheckAU}(\mathit{CLTS}, \mu, \chi, \phi, \chi', \phi', \zeta)$
    ...
**endsw**
```

Fig. 4 Backbone of the model checking algorithm.

6.1 Model checking EU formulae

We explain the idea of the subcall $\mathit{CheckEU}$ for the $\mathbf{EU}$ operator. A ϕ-state satisfies a $\mathbf{E}(\phi\,_{\chi}\mathbf{U}_{\chi'}\,\phi')$ under ζ if it has a ζ-path that consists of (χ, ϕ)-transitions until a (χ', ϕ')-transition is performed. Our model checking algorithm is based on the CTL model checking algorithm for $\mathbf{EU}$ [8], where the analysis starts from ϕ'-states and proceeds in a backward fashion over ϕ-states.

First, recursively, the states are determined that satisfy ϕ as well as those that satisfy ϕ'. Then, ϕ-states with an outgoing (χ', ϕ')-transition which conforms to ζ are searched for. Later, we move backward over (χ, ϕ)-transitions which conform to ζ in a breadth-first fashion. Backward movement is stopped when no new ϕ-state can be explored. All ϕ-states visited in the backward analysis satisfy $\mathbf{E}(\phi\,_{\chi}\mathbf{U}_{\chi'}\,\phi')$ under ζ. The pseudocode of the algorithm is given in Fig. 5.

The FIFO queue *explore* initially contains the states where the backward analysis starts from: ϕ-states with an outgoing (χ', ϕ')-transition which conforms to ζ.

```
**Procedure** $\mathit{CheckEU}(\mathit{CLTS}, \chi, \phi, \chi', \phi', \zeta)$
**Output**: The states that satisfy $\mathbf{E}(\phi\, {}_{\chi}\mathbf{U}_{\chi'}\, \phi')$ under $\zeta$ in $\mathit{CLTS} \equiv \langle S, \Lambda, \rightarrow, s_0\rangle$
$T_\phi := \mathit{ModelCheck}(\mathit{CLTS}, \phi, \zeta)$;
$T_{\phi'} := \mathit{ModelCheck}(\mathit{CLTS}, \phi', \zeta)$;
$\mathit{explore} := \{s \in T_\phi \mid (s, (\mathcal{C}, \eta), t) \in \rightarrow,\ t \in T_{\phi'},\ \eta \models \chi',\ \zeta \cap \neg\mathcal{C} = \emptyset\}$;
$\mathit{visited} := \emptyset$;
$\rightarrow' := \{(t, (\mathcal{C}, \eta), t') \in \rightarrow \mid t, t' \in T_\phi,\ \eta \models \chi,\ \zeta \cap \neg\mathcal{C} = \emptyset\}$;
**while** $\mathit{explore} \neq \emptyset$ **do**
    $t_0 := \mathit{dequeue}(\mathit{explore})$;
    $\mathit{visited} := \mathit{visited} \cup \{t_0\}$;
    **for all** $(s, (\mathcal{C}, \eta), t_0) \in \rightarrow'$ **do**
        **if** $s \notin \mathit{explore} \cup \mathit{visited}$ **then** $\mathit{enqueue}(s, \mathit{explore})$

**return** $\mathit{visited}$;
```

Fig. 5 Model checking **EU** formulae.

Backward movement over (χ, ϕ)-transitions that conform to ζ is assisted by initially finding such transitions, denoted by transition relation $\rightarrow'$. State visited during backward movement are stored in the set $\mathit{visited}$. The correctness of the algorithm is implied by the fact that $\forall s \in \mathit{explore}\ (s \models_\zeta \mathbf{E}(\phi\, {}_{\chi}\mathbf{U}_{\chi'}\, \phi'))$.

As an example we verify $\mathbf{E}(\mathit{true}\, {}_{a\vee\tau}\mathbf{U}_b\, \mathbf{E}(\mathit{true}\, {}_{a\vee\tau}\mathbf{U}_c\, \mathit{true}))$ under $\{\}$ over the CLTS given in Fig. 6. First the states satisfying $\mathbf{E}(\mathit{true}\, {}_{a\vee\tau}\mathbf{U}_c\, \mathit{true})$ are found. The breadth-first backward search starts from s_7, as this is the only state with an outgoing c-transition. The transitions from s_4 to s_6 and from s_7 to s_9 are removed because they do not carry an action from $\{a, \tau\}$. The verification of the inner **EU** formula ends with $\mathit{visited} = \{s_6, s_7\}$. To verify the outer **EU** formula, the breadth-first backward analysis starts from s_4, because from this state there is a b-transition to s_6 where the inner **EU** formula holds. Again the transitions from s_4 to s_6 and from s_7 to s_9 are removed. The search computes $\mathit{visited} = \{s_0, s_1, s_2, s_4, s_5\}$, which is returned as the result of the verification of the outer **EU** formula.

$$\begin{tikzcd}
s_0 \arrow[r, "(\{\}, a)"] & s_1 \arrow[r, "\left(\left\{\begin{matrix} D \rightsquigarrow C, \\ D \not\rightsquigarrow A, B \end{matrix}\right\}, \tau\right)"] & s_2 \arrow[r, "\left(\left\{\begin{matrix} C \rightsquigarrow A, D, \\ C \not\rightsquigarrow B \end{matrix}\right\}, \tau\right)"] \arrow[dr, "\left(\left\{\begin{matrix} C \rightsquigarrow A, B, \\ C \not\rightsquigarrow D \end{matrix}\right\}, \tau\right)"] & s_3 \arrow[r, bend left, "\left(\left\{\begin{matrix} A \rightsquigarrow B, \\ A \not\rightsquigarrow C, D \end{matrix}\right\}, \tau\right)"] & s_{10} \arrow[l, bend left, "(\{B \rightsquigarrow A, C, D\}, \tau)"] & s_7 \arrow[r, "(\{\}, c)"] & s_9 \\
 & & s_5 \arrow[r, "\left(\left\{\begin{matrix} D \rightsquigarrow A, B, \\ D \not\rightsquigarrow C \end{matrix}\right\}, \tau\right)"'] & s_4 \arrow[r, "(\{\}, b)"] & s_6 \arrow[r, "(\{B \not\rightsquigarrow A, C, D\}, \tau)"'] \arrow[ur, "\left(\left\{\begin{matrix} B \rightsquigarrow C, \\ B \not\rightsquigarrow A, D \end{matrix}\right\}, \tau\right)"] & s_8 &
\end{tikzcd}$$

Fig. 6 The CLTS to be checked for $\mathbf{E}(\mathit{true}\, {}_{a\vee\tau}\mathbf{U}_b\, \mathbf{E}(\mathit{true}\, {}_{a\vee\tau}\mathbf{U}_c\, \mathit{true}))$.

The worst-case time and memory complexity to check an **EU** formula is $O(V + E)$, where V and E are the number of states and transitions of the underlying CLTS. By contrast, model checking such a CTL formula using the standard algorithm with the approach of [13], where the topology is modeled as a part of states, has a worst-case time and memory complexity of $O(4^{\frac{n^2-n}{2}} \times (V+E))$, where n is the number of nodes; 4 originates from the four types of links, and $\frac{n^2-n}{2}$ is the number of possible links among the n nodes. We note however that our model checking algorithm for $\mathbf{AU}$ formulae, which will be explained in the next section, faces a similar exponential blow-up.

A bottleneck for the performance can be the check whether $s \notin \mathit{explore} \cup \mathit{visited}$. In general the generated state space is so large that the bulk of it has to be stored on disk, and disk accesses to perform the aforementioned check are very expensive. In line with [22], this overhead can be alleviated considerably by exploiting a Bloom filter to reduce the number of redundant disk accesses in cases where the state s was not yet generated, i.e., is not present on disk.

6.2 Model checking $\mathbf{AU}$ formulae

The procedure $\mathit{CheckAU}$ model checks the $\mathbf{AU}$ operator. In general a ϕ-state satisfies a formula $\mathbf{A}^{\mu}(\phi\,{}_{\chi}\mathbf{U}_{\chi'}\,\phi')$ under ζ if all its ζ-paths consist of (χ, ϕ)-transitions until either a (χ', ϕ')-transition is performed or the topology formula μ is violated. Paths that violate the $\mathbf{AU}$ formula can be searched for with a backward analysis in a depth-first fashion. A state t may be visited for different values of network constraints, gathered over different paths. The backward analysis should be redone for each network constraint $\mathcal{C}$, except when t was visited earlier for a network constraint $\mathcal{C}'$ with $\mathcal{C}' \subseteq \mathcal{C}$.

We start the backward search from ϕ-states that violate the $\mathbf{AU}$ formula because they have either (1) an outgoing transition (which conforms to ζ) that is neither a (χ', ϕ')- nor a (χ, ϕ)-transition, or (2) no outgoing transition, or (3) an infinite (χ, ϕ)-path that does not violate μ (and conforms to ζ). All preceding states on backward (χ, ϕ)-paths that do not violate μ (with transitions that conform to ζ) violate $\mathbf{A}^{\mu}(\phi\,{}_{\chi}\mathbf{U}_{\chi'}\,\phi')$. Furthermore, $\neg\phi$-states trivially violate this formula.

For the sake of efficiently finding ϕ-states of type (3), we assume *strong fairness*, meaning that on any infinite path each infinitely often enabled transition is infinitely often executed. Considering only (χ, ϕ)-transitions that conform to ζ, we determine *terminal* strongly connected components (SCCs) (i.e., SCCs from which no other SCC can be reached). Owing to strong fairness, each infinite (χ, ϕ)-path is guaranteed to end up in such a terminal SCC, and traverse each transition in this SCC. We check for each terminal SCC whether its accumulated network constraints do not violate μ.

The pseudocode of the algorithm is given in Fig. 7. The procedure Clean removes all transitions that do not conform to ζ, and turns states that can perform a transition (which conforms to ζ) that is neither a (χ', ϕ')- nor a (χ, ϕ)-transition into deadlock states, by removing all their outgoing transitions. This way ϕ-states of type (1) are cast to type (2).

The procedure TSCC computes the terminal SCCs of (χ, ϕ)-transitions, and in parallel collects the network constraints on the transitions in each such terminal SCC. It returns a function $\mathit{constraint}$ that maps each state in a terminal SCC of (χ, ϕ)-transitions to the accumulation of all network constraints on the transitions in this terminal SCC; states that are not in such a terminal SCC are mapped to the empty

```
**Procedure** $\mathit{CheckAU}(\mathit{CLTS}, \mu, \chi, \phi, \chi', \phi', \zeta)$
**Output**: The states that satisfy $\mathbf{A}^{\mu}(\phi\,{}_{\chi}\mathbf{U}_{\chi'}\,\phi')$ under $\zeta$ in $\mathit{CLTS} \equiv \langle S, \Lambda, \rightarrow, s_0 \rangle$
$T_\phi := \mathit{ModelCheck}(\mathit{CLTS}, \phi, \zeta)$;
$T_{\phi'} := \mathit{ModelCheck}(\mathit{CLTS}, \phi', \zeta)$;
$\rightarrow' := \mathit{Clean}(\mathit{CLTS}, \chi, T_\phi, \chi', T_{\phi'}, \zeta)$;
$\mathit{constraint} := \mathit{TSCC}(\rightarrow', \chi, T_\phi)$;
$\mathit{start} := \mathit{Start}(\rightarrow', \mu, \chi, T_\phi, \chi', T_{\phi'}, \mathit{constraint})$;
$\mathit{violate} := \mathit{start}$;
$\mathit{visited} := \{t \mapsto \{\gamma \in \mu \mid \mathcal{C}^{+}_{\Gamma}(\gamma) \cap \neg\mathit{constraint}(t) = \emptyset\} \mid t \in \mathit{start}\} \cup \{(t \mapsto \emptyset) \mid t \notin \mathit{start}\}$;
**while** $\mathit{start} \neq \emptyset$ **do**
    choose a $t_0 \in \mathit{start}$;
    $\mathit{start} := \mathit{start} \setminus \{t_0\}$;
    $\mathit{explore} := \{(t_0, \{\gamma \in \mu \mid \mathcal{C}^{+}_{\Gamma}(\gamma) \cap \neg\mathit{constraint}(t) = \emptyset\})\}$;
    **while** $\mathit{explore} \neq \emptyset$ **do**
        $(t_1, \Gamma_1) := \mathit{pop}(\mathit{explore})$;
        $\mathit{violate} := \mathit{violate} \cup \{t_1\}$;
        **for all** $(t_2, (\mathcal{C}_2, \eta), t_1) \in \rightarrow'$ **do**
            $\Gamma := \{\gamma \in \Gamma_1 \mid \mathcal{C}^{+}_{\Gamma}(\gamma) \cap \neg\mathcal{C}_2 = \emptyset\}$;
            **if** $\Gamma \nsubseteq \mathit{visited}(t_2) \wedge \Gamma \neq \emptyset$ **then**
                $\mathit{push}((t_2, \Gamma \setminus \mathit{visited}(t_2)), \mathit{explore})$;
                $\mathit{upd}(\mathit{visited}, t_2, \Gamma)$;

**return** $T_\phi \setminus \mathit{violate}$;
```

Fig. 7 Model checking $\mathbf{AU}$ formulae.

network constraint. Next the procedure *Start* returns the set *start* of states that are in a terminal SCC of (χ, ϕ)-transitions in which (I) none of the states can perform a (χ', ϕ')-transition, and (II) the accumulation of all network constraints on the transitions do not violate μ.

The backward movement, which every time starts from a state in *start*, is performed in a depth-first fashion over (χ, ϕ)-transitions which conform to ζ. During this backward movement, network constraints on transitions are accumulated. The backward search proceeds as long as the accumulated network constraints do not violate μ. Visited states violate the $\mathbf{AU}$ formula; they are stored in the set *violate*, which initially consists of the states in *start*. We remark that the accumulated network constraint $\mathcal{C}_1 \cup \mathcal{C}_2$ does not violate μ, implied by $\exists\gamma \in \mu\,(\mathcal{C}^{+}_{\Gamma}(\gamma) \cap \neg(\mathcal{C}_1 \cup \mathcal{C}_2) = \emptyset)$, if and only if $\exists\gamma \in \{\gamma' \in \mu \mid \mathcal{C}^{+}_{\Gamma}(\gamma') \cap \neg\mathcal{C}_1 = \emptyset\}\,(\mathcal{C}^{+}_{\Gamma}(\gamma) \cap \neg\mathcal{C}_2 = \emptyset)$. Hence, instead of accumulating network constraints over the paths and then checking if they do not violate μ, the topologies which lead to the satisfaction of μ are initially computed and refined while passing a transition. Therefore, the function $\mathit{visited}$ maps each visited state to the set of topologies for which the backward analysis was performed; states that have not yet been visited are mapped to the empty set. Hence, the backward search proceeds as long as the set of topologies (which makes μ holds over the path) is not empty. Furthermore, the backward analysis is redone for each visited state s with the set of topologies Γ if $\Gamma \nsubseteq \mathit{visited}(s)$.

Each backward search is coordinated by means of the stack *explore*, which initially consists of the state t_0 in start where the search starts, paired with topologies that make μ holds with respect to the accumulation $constraint(t_0)$ of network constraints in the terminal SCC of t_0. Repeatedly the top $(t_1, \varGamma_1)$ of $explore$ is popped, and t_1 is added to $violate$. Furthermore, for each incoming transition $(t_2, (\mathcal{C}_2, \eta), t_1)$ of t_1, $\varGamma_1$ is refined into $\varGamma$ with respect to $\mathcal{C}_2$ to denote the topologies for which the state t_2 has been visited. Then, it is checked whether state t_2 has been visited for a new topology. If this is the case, then the pair $(t_2, \varGamma \setminus visited(s))$ is added to the stack $explore$, where $\varGamma \setminus visited(s)$ is the set of newly visited topologies for which the backward analysis should be redone. Furthermore, $visited$ is updated by the new topologies for t_2, denoted by $upd(visited, t_2, \varGamma)$.

Finally, when the backward search has been performed exhaustively, meaning that *start* has become empty, all states in T_ϕ that are not in *violate* satisfy $\mathbf{A}^{\mu}(\phi\,_{\chi}\mathbf{U}_{\chi'}\,\phi')$ under ζ.

The correctness of our algorithm is established based on the following assertions, which are straightforward to prove:

1. The states on the $explore$ stack form a (χ, ϕ)-path in the semantic model that conforms to ζ;
2. $\forall (s, \varGamma) \in explore\, (\varGamma \neq \emptyset)$.
3. The sets of topologies on the stack constitute a chain, i.e., for any two consecutive elements $(s_1, \varGamma_1)$ directly above $(s_2, \varGamma_2)$ on the stack, $\varGamma_1 \subseteq \varGamma_2$ and s_1 is a descendant of s_2.

For each $s \in violate$, there was $(s, \varGamma)$ on the top of the stack before s was being added to $violate$. The first assertion implies that there exists a (χ, ϕ)-path that conforms to ζ from s to the state at the bottom of the stack, which belongs to $start$. The second assertion implies that $\varGamma \neq \emptyset$. Therefore, the third assertion implies that the accumulated network constraint over this path does not violate μ. Hence, s indeed violates $\mathbf{A}^{\mu}(\phi\,_{\chi}\mathbf{U}_{\chi'}\,\phi')$ under ζ.

Our model checking algorithm is inspired by the CTL model checking algorithm for $\mathbf{EG}\phi$ formulae [9], where the backward search starts from SCCs with ϕ-states, i.e., states satisfying $\mathbf{EG}\phi$. We remark that the states in our terminal SCCs violate $\mathbf{AU}$. Furthermore, the classical CTL model checker for $\mathbf{AU}$ formulae is based on a forward analysis in a depth-first fashion; a state satisfies $\mathbf{AU}$ if none of its successors violates $\mathbf{AU}$ [8]. However, fairness conditions cannot be addressed during the forward analysis. To this end, the framework is extended with predicates over states. Then states with a fair-path, i.e., a path over which each fairness predicate is infinitely many times satisfied, are labeled with the auxiliary predicate Q (through a backward analysis from fair-SCCs, i.e., SCCs with at least one state satisfying each fairness predicate). The $\mathbf{AU}$ formulae with fairness conditions are found using the $\mathbf{EU}$- and $\mathbf{EG}$-algorithms. The former considers Q-labels while the latter starts the backward search from fair-SCCs with ϕ-states.

Each state is in the worst case visited for all possible accumulated network constraints. If the set *Loc* contains n addresses, there are $\binom{n-2}{2}$ (dis)connectivity pairs, and consequently the number of possible accumulated network constraints is 2^{n^2-n}. Hence, each state is visited at most 2^{n^2-n} times. This means model checking of $\mathbf{AU}$ formulae is only feasible in case of a small number of addresses. However, since interesting mobility scenarios typically require only few addresses, this is not a serious limitation of our approach.

The time complexity of $\mathbf{AU}$ model checking is $O(2^{n^2-n} \times (V+E))$, with memory usage $O(V+E)$. Since the time and space complexities are linear with respect to the number of states and transitions, model checking MANET protocols with arbitrary mobility is feasible. We recall that model checking CTL formula over semantic models where topology is modeled as a part of states, following the approach of [13], has a time complexity of $O(2^{n^2-n} \times (V+E))$ with memory usage $O(2^{n^2-n} \times (V+E))$. Note that we have not considered the memory usage of $visited$ in favor of the space needed to store topologies as part of states in the latter approach.

As an example, we check the formula $\mathbf{A}^{\,D \dashrightarrow B}(true_{\;a\vee\tau}\mathbf{U}_b\mathbf{E}(true_{\;a\vee\tau}\mathbf{U}_c\,true))$ under $\{\}$ over the CLTS given in Fig. 6, using our model checking algorithm. First transitions from s_4 to s_6 and from s_7 to s_9 are removed because they do not carry an action from $\{a,\tau\}$. The terminal (χ,ϕ)-SCCs of which the accumulated network constraints do not violate $D \dashrightarrow B$, found by procedure $FindSCC$, are $\{s_3, s_{10}\}$, $\{s_8\}$ and $\{s_9\}$. Therefore, they are collapsed as shown in Fig. 8, while $visited(SCC_0) = \{\{A \rightsquigarrow B,\ A \not\rightsquigarrow C, D,\ B \rightsquigarrow A, C, D\}\}$, $visited(SCC_1) = \{\{\}\}$, and $visited(SCC_2) = \{\{\}\}$. Furthermore, the relation SCC is set to $\{(SCC_0, \{s_3, s_{10}\}), (SCC_1, \{\}), (SCC_2, \{\})\}$. The set end contains $\{s_7, SCC_0, SCC_1, SCC_2\}$, since s_7 contains a transition which is neither from $\{a,\tau\}$ nor from $\{b\}$. The backward analysis starts from SCC_0 for $head(visited(SCC_0))$. Upon movement to s_2, the accumulated network constraints are $\{A \rightsquigarrow B,\ A \not\rightsquigarrow C, D,\ B \rightsquigarrow A, C, D,\ C \rightsquigarrow A, D,\ C \not\rightsquigarrow B\}$ are valid for $D \dashrightarrow B$, and consequently it is added to $visited(s_2)$. The backward analysis proceeds to s_0, while $temp = \{SCC_0, s_2, s_1, s_0\}$. The backward analysis from states SCC_1 and s_7 stops at state s_6, while $visited(s_6) = \{\{B \rightsquigarrow C,\ B \not\rightsquigarrow A, D\}, \{B \not\rightsquigarrow A, C, D\}\}$ and states SCC_1, s_7, and s_6 are added to $temp$. The backward analysis from SCC_2 stops immediately, while it is added to $temp$. Consequently states $\{s_4, s_5\}$ are returned by the algorithm. Note that the path $s_0(\{\}, a)s_1(\{D \not\rightsquigarrow A, B,\ D \rightsquigarrow C\}, \tau)s_2(\{C \rightsquigarrow A, D,\ C \not\rightsquigarrow B\}, \tau)s_3(\{A \rightsquigarrow B,\ A \not\rightsquigarrow C, D\}, \tau)s_{10}$ is a valid path for the multi-hop constraint $D \dashrightarrow B$ that does not end with a $(b, \mathbf{E}(true_{\;a\vee\tau}\mathbf{U}_{B \dashrightarrow C}\,ctrue))$-transition.

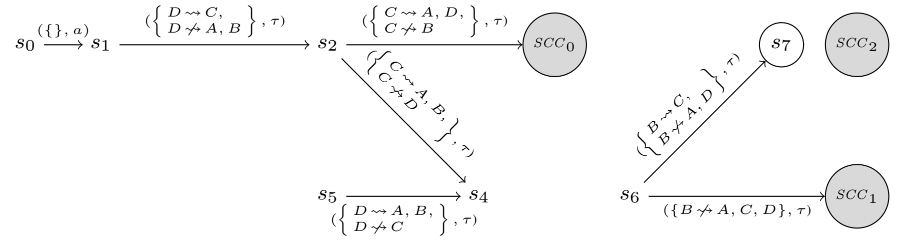

s_0
$(\{\}, a)$
s_1
$(\{D \rightsquigarrow C,\ D \not\rightsquigarrow A, B\}, \tau)$
s_2
$(\{C \rightsquigarrow A, D,\ C \not\rightsquigarrow B\}, \tau)$
SCC_0
$(\{C \rightsquigarrow A, B,\ C \not\rightsquigarrow D\}, \tau)$
s_5
s_4
$(\{D \rightsquigarrow A, B,\ D \not\rightsquigarrow C\}, \tau)$
s_7
SCC_2
$(\{B \rightsquigarrow C,\ B \not\rightsquigarrow A, D\}, \tau)$
s_6
$(\{B \not\rightsquigarrow A, C, D\}, \tau)$
SCC_1

Fig. 8 The CLTS used during backward analysis to check $\mathbf{A}^{\,D \dashrightarrow B}(true_{\;a\vee\tau}\mathbf{U}_b\,\mathbf{E}(true_{\;a\vee\tau}\mathbf{U}_c\,true))$: the white and gray states are of the first and second types respectively.

6.3 Model checking **EW** formulae

Formula $\mathbf{E}(\phi\,_\chi\mathbf{W}_{\chi'}\,\phi')$ is satisfied by ϕ-states that (1) either have a ζ-path of (χ, ϕ)-transitions that end with a (χ', ϕ')-transition, or (2) have a (χ, ϕ) ζ-path.

States of first type are found by calling procedure $\mathit{CheckEU}$. To find states of second type, terminal SCCs made of (χ, ϕ)-transitions are search for. Preceding states over (χ, ϕ)-transitions that conform to ζ satisfy $\mathbf{E}(\phi\,_\chi\mathbf{W}_{\chi'}\,\phi')$. To this aim, CLTS is restricted to ϕ-states that were not found by $\mathit{CheckEU}$ and χ-transitions which conform to ζ. The resulting CLTS is decomposed into SCCs, while each SCC is collapsed into a single state. A backward analysis in a breath-first fashion is performed from each terminal state. States visited during backward analysis satisfy $\mathbf{E}(\phi\,_\chi\mathbf{W}_{\chi'}\,\phi')$ under ζ and so do states within visited collapsed SCCs.

```
**Procedure** $\mathit{CheckEW}(\mathit{CLTS}, \phi, \chi, \chi', \phi', \zeta)$
**Output**: The states that satisfy $\mathbf{E}(\phi\,_\chi\mathbf{W}_{\chi'}\,\phi')$ under $\zeta$ in $\mathit{CLTS} \equiv \langle S, \Lambda, \rightarrow, s_0\rangle$
$T_\phi := \mathit{ModelCheck}(\mathit{CLTS}, \phi, \zeta)$;
$T_{\phi'} := \mathit{ModelCheck}(\mathit{CLTS}, \phi', \zeta)$;
$\mathit{visited} := \mathit{CheckEU}(\mathit{CLTS}, \phi, \chi, \chi', \phi', \zeta)$;
$\rightarrow' := \{(t, (\mathcal{C}, \eta), t') \in \rightarrow \mid t, t' \in T_\phi \setminus \mathit{visited},\ \eta \models \chi,\ \zeta \cap \neg\mathcal{C} = \emptyset\}$;
$\rightarrow'' := \mathit{CollapSCC}(\rightarrow')$;
$\mathit{explore} := \{s \mid s$ has no outgoing transition $\in\rightarrow''\}$;
**while** $\mathit{explore} \neq \emptyset$ **do**
    $t_0 := \mathit{head}(\mathit{explore})$;
    $\mathit{visited} := \mathit{visited} \cup \{t_0\}$;
    $\mathit{dequeue}(\mathit{explore})$;
    **for all** $(s, (\mathcal{C}, \eta), t_0) \in \rightarrow''$ **do**
        **if** $s \notin \mathit{explore} \cup \mathit{visited}$ **then** $\mathit{enqueue}(s, \mathit{explore})$

$\mathit{result} := \{t \in T_\phi \mid t \in \mathit{visited} \vee \exists s\,(t \in \mathit{SCC}(s))\}$;
**return** $\mathit{result}$;
```

Fig. 9 Model checking **EW** formulae.

The pseudocode of the algorithm is given in Fig. 9. The procedure $\mathit{CollapSCC}$ (1) computes the SCCs of a CLTS, (2) collapses each SCC to a single state, (3) adds a mapping from each (collapsed) SCC to its states in the relation SCC. The FIFO queue $\mathit{explore}$ initially contains the states where the backward analysis starts from: ϕ-states with a (χ, ϕ)-path whose transitions conform to ζ. State visited during backward movement are stored in the set $\mathit{visited}$. A ϕ-states satisfies $\mathbf{E}(\phi\,_\chi\mathbf{W}_{\chi'}\,\phi')$ if either is in $\mathit{visited}$ or within an SCC. The correctness of the algorithm is implied by the correctness of the **EU** model checking algorithm and the fact that $\forall s \in \mathit{explore}\ (s \models_\zeta \mathbf{E}(\phi\,_\chi\mathbf{W}_{\chi'}\,\phi'))$.

The time complexity to check an **EW** formula is $O(V + E)$. However model checking CTL formula over semantic models where topology is modeled as a part of states, following the approach of [13], has a time complexity of $O(2^{n^2-n} \times (V+E))$.

6.4 Model checking $\mathbf{AW}$ formulae

Model checking $\mathbf{AW}$ is similar to $\mathbf{AU}$. In general a ϕ-state satisfies a formula $\mathbf{A}^{\mu}(\phi\,{}_{\chi}\mathbf{W}_{\chi'}\,\phi')$ under ζ if all its ζ-paths consist of (χ, ϕ)-transitions as long as either a (χ', ϕ')-transition is performed or the topology formula μ is violated. A ϕ-state trivially violates this property if it has an outgoing transition (which conforms to ζ) that is neither a (χ', ϕ')- nor a (χ, ϕ)-transition. Preceding states over a (χ, ϕ)-path that does not violate μ, and its transitions conform to ζ, violate $\mathbf{A}^{\mu}(\phi\,{}_{\chi}\mathbf{W}_{\chi'}\,\phi')$ and so do $\neg\phi$-states. Such states can be searched similar to $\mathbf{AU}$.

7 Protocol Analysis with CACTL

CACTL can be used to express and verify interesting properties of MANET protocols. For example, the most fundamental error in routing protocol operations is failure to route correctly. The correct operation of MANET routing protocols is defined as follows [34]: "*If from some point in time on there exists a path between two nodes, then the protocol must be able to find some path between the nodes. Furthermore, when a path has been found, and for the time it stays valid, it must be possible to send packets along the multi-hop connection from the source node to the destination node*". To specify such a property, let $init(src, dst)$ indicate initialization of sending data from a node with address src to a specific address dst, while $get(dst)$ indicates receipt of this data at the destination. Note that a request to send data may initialize a route discovery in cases where there is no route from the sender to the destination. The property "whenever there exists a multi-hop connection from src to dst and from dst to src which is valid for a sufficiently long period of time, each $init(src, dst)$ is eventually proceeded by $get(dst)$", for $scr = A$ and $dst = B$, is specified by the CACTL formula

$$\mathbf{A}^{true}(true\,{}_{\neg init(A,B)}\mathbf{W}_{init(A,B)}\,\mathbf{A}^{A \dashrightarrow B \wedge B \dashrightarrow A}(true\,{}_{\tau}\mathbf{U}_{get(B)}\,true))$$

where τ abstracts away from communications between nodes. By model checking the CLTS model of a MANET in which the nodes deploy a routing protocol, we can verify this property with respect to arbitrary topology changes. We verified this property for the new version of AODVv2 (version 11),[2] modeled in an extended version of Rebeca [32], called wRebeca [36] and supported by a tool [2], which is an actor-based modeling language [1]. We thus detected a scenario in which the property is violated in favor of loop avoidance. This property was examined for the AODV protocol using CTL in [13].

7.1 Verification of a leader election protocol

To illustrate the effectiveness of CACTL in the analysis of MANETs, we specify and analyze a leader election protocol for MANETs introduced in [33]. To this aim,

[2] Perkins, C., Ratliff, S., Dowdell, J., Steenbrink, L., Mercieca, V.: Ad hoc on-demand distance vector (AODVv2) routing, `https://tools.ietf.org/html/draft-ietf-manet-aodvv2-11`.

RRBPT is extended with abstract data types in the same way as the specification language μCRL [4,14], and its actions and process names are parameterized with data, following the approach of [18]. We add two operators that intertwine processes with data: *sum* to let receive actions range over the data values of their message parameters (to be able to receive any value from the environment), and *condition* to let behavior depend on data values. The process $\sum_{d:D} t$ is equivalent to $t[u_1/d] + t[u_2/d] + \ldots$ for any values $u_1, u_2, \ldots$ over the data domain D. The process $[b]t_1 \diamond t_2$ behaves as t_1 if b evaluates to true, and otherwise as t_2.

7.1.1 Protocol specification

Leader election algorithms aim at electing a unique leader from a fixed set of nodes. In the context of MANETs, such protocols must consider arbitrary topology changes and aim at finding a unique leader within a strongly connected component (SCC) [33].

The algorithm operates by first "growing" and then "shrinking" a spanning tree rooted at the source node that initiates the election algorithm, using the so-called echo algorithm [6,15]. After the spanning tree has shrunk completely, the source node will have the required information to elect the leader. The spanning tree is constructed by broadcasting an $\mathit{election}$ message from the source node to the leaf nodes. The parent of each node in the spanning tree is the node from whom it receives the $\mathit{election}$ message for the first time. After receiving the $\mathit{election}$ message, the node broadcasts $\mathit{election}$ to its other neighbors. The spanning tree is shrunk by broadcasting ack message from the leaf nodes to the source node. Each node after gathering ack messages from all its children, determines the most-valued node in its subtree, and sends this information to its parent through an ack message. Since nodes may crash, each node keeps track of the status of children from whom it should still receive an ack, by periodically sending and receiving probe and reply messages. The source node can determine the leader when it has gathered ack messages from all its children. It announces the leader by means of a leader message, which is forwarded down the spanning tree.

Due to node crashes and mobility of nodes, a spanning tree may be partitioned or two spanning trees may merge. A node triggers a fresh leader election process when it gets disconnected from its leader or restarts from the crash state. Thus more than one node can initiate the leader election process independently, leading to concurrent computations. In [33], concurrent computations are handled by requiring that at any time each node participates in at most one computation. To achieve this, each spanning tree is identified by a pair of its source node identifier id and a sequence number num, which is incremented each time a node starts a fresh computation, called *computation index*, and all messages except leader are tagged with $\langle \mathit{id}, \mathit{num} \rangle$. A total order is defined on computation indices. Each node belongs to a spanning tree from which it received the $\mathit{election}$ message with the highest computation index. When two disjoint MANETs connect, the algorithm allows the MANETS to terminate their ongoing computations and then exchange their leader through leader messages; the combined MANET adopts the leader with the higher value.

In the specification and verification of the protocol, we abstracted away exchanges of probe and reply messages: each node may non-deterministically conclude that it has received ack messages from all its children and stop processing subsequent ack messages. Furthermore, with the help of the sensing operator, each node adapts its behavior in case it gets disconnected from its parent. To make the state space finite we assume that each node can disconnect from its leader only once. We define a total order on the network address of nodes and for simplicity assume that the value of a node is the same as its network address.

Our specification uses as abstract data types the Booleans $Bool$ (with domain values T and F), addresses Loc, natural numbers Nat, and computation indices $CompInd$. The variables maintained by each node are: $elec$ and ack of type $Bool$, where $elec$ is true while the node is involved in a computation, and ack is true if the node has not yet sent an ack message to its parent; lid, max and $parent$ of type Loc, where lid denotes the address of the supposed leader (which is updated when the node receives a $leader$ message), max is the highest value the node has encountered in a computation, and $parent$ the address of the node's parent in the spanning tree; src defines the computation index of the spanning tree of which the node currently is a member. Values of $CompInd$ are pairs of a sequence number and an address, paired by the constructor function dc. To access the items of src, for the sake of readability, we use a dot notation like $src.id$; num of type Nat denotes the node's sequence number.

The specification of the algorithm is given in Fig. 10. The pre-conditions to send and receive $election$, ack and $leader$ messages and parameter updates match with the specification given in [33] with two minor differences. We elaborate on each summand of the specification, and highlight points where our specification differs from [33].

In the first summand, in case a node has a leader different from itself (denoted by $(id \neq lid) \wedge (lid \neq ?)$) and is disconnected from its leader (examined by the sense operator), the node initiates the election algorithm by broadcasting a leader message. The conjunct "$num < 1$" prevents an infinite growth of the state space due to an increase of num values within computation indices. An election process is also initiated, as indicated by the second summand, when a node has no leader and is not already participating in an election process (i.e., $\neg elec \wedge (lid = ?)$). This is the case when a node restarts from the crash state. After sending the message $election$, $elec$ and ack are set to true, while max is set to the node's identifier. Furthermore, $parent$ and lid are set to unknown, while src is set to $dc(num, id)$.

The third summand specifies how a node handles received $election$ messages. Such a message is processed if the receiving node has not sent its ack yet and the message belongs to an election process with a higher computation index than what the node has seen so far (i.e., $elec \wedge dc(num1, id1) > src \wedge ack$), or the node has just recovered from a crash and hence has no leader and has not participated in an election process yet (i.e., $\neg elec \wedge (lid = ?)$). In those cases the node joins the election process, by setting its src to $dc(num1, id1)$, and setting its parent to the node $id2$ it received the message from.

The fourth summand specifies when a node sends its ack message. A non-root node that has not sent its ack message before (indicated by $(src.id \neq id) \wedge ack$) announces the maximum identifier max it knows by sending the message $ack(src, max)$,

$$
\begin{array}{l}
\mathit{node}(\mathit{id}:\mathit{Loc}, \mathit{elec}:\mathit{Bool}, \mathit{ack}:\mathit{Bool}, \mathit{lid}:\mathit{Loc}, \mathit{max}:\mathit{Loc}, \mathit{parent}:\mathit{Loc}, \mathit{src}:\mathit{CompInd},\\
\qquad \mathit{num}:\mathit{Nat}) \overset{def}{=}\\[1ex]
\quad [(\mathit{id} \neq \mathit{lid}) \wedge (\mathit{lid} \neq ?) \wedge \mathit{num} < 1]\\
\qquad \mathit{sense}(\mathit{lid}, \mathit{node}(\mathit{id}, \mathit{elec}, \mathit{ack}, \mathit{lid}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num}),\\
\qquad\quad \mathit{snd}(\mathit{election}(\mathit{dc}(\mathit{num}, \mathit{id}), \mathit{id})).\mathit{node}(\mathit{id}, T, T, ?, \mathit{id}, ?, \mathit{dc}(\mathit{num}, \mathit{id}), \mathit{num}+1)) \diamond 0\ +\\[1ex]
\quad [\neg \mathit{elec} \wedge (\mathit{lid} = ?) \wedge \mathit{num} < 1]\\
\qquad \mathit{snd}(\mathit{election}(\mathit{dc}(\mathit{num}, \mathit{id}), \mathit{id})).\mathit{node}(\mathit{id}, T, T, ?, \mathit{id}, ?, \mathit{dc}(\mathit{num}, \mathit{id}), \mathit{num}+1) \diamond 0\ +\\[1ex]
\quad \sum_{\mathit{num1}:\mathit{Nat}} \sum_{\mathit{id1}:\mathit{Loc}} \sum_{\mathit{id2}:\mathit{Loc}} \mathit{rcv}(\mathit{election}(\mathit{dc}(\mathit{num1}, \mathit{id1}), \mathit{id2})).\\
\qquad [(\neg \mathit{elec} \wedge (\mathit{lid} = ?)) \vee (\mathit{elec} \wedge \mathit{dc}(\mathit{num1}, \mathit{id1}) > \mathit{src} \wedge \mathit{ack})]\\
\qquad\quad \mathit{snd}(\mathit{election}(\mathit{dc}(\mathit{num1}, \mathit{id1}), \mathit{id})).\mathit{node}(\mathit{id}, T, T, \mathit{lid}, \mathit{id}, \mathit{id2}, \mathit{dc}(\mathit{num1}, \mathit{id1}), \mathit{num}) \diamond\\
\qquad\quad \mathit{node}(\mathit{id}, \mathit{elec}, \mathit{ack}, \mathit{lid}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num})\ +\\[1ex]
\quad [\mathit{ack} \wedge (\mathit{src}.\mathit{id} \neq \mathit{id})](\mathit{snd}(\mathit{ack}(\mathit{src}, \mathit{max})).\mathit{node}(\mathit{id}, \mathit{elec}, F, \mathit{lid}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num}))) \diamond 0\ +\\[1ex]
\quad \sum_{\mathit{mid}:\mathit{Loc}} \mathit{rcv}(\mathit{ack}(\mathit{src}, \mathit{mid})).[\mathit{ack} \wedge (\mathit{mid} > \mathit{max})]\\
\qquad \mathit{node}(\mathit{id}, \mathit{elec}, \mathit{ack}, \mathit{lid}, \mathit{mid}, \mathit{parent}, \mathit{src}, \mathit{num}) \diamond\\
\qquad \mathit{node}(\mathit{id}, \mathit{elec}, \mathit{ack}, \mathit{lid}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num})\ +\\[1ex]
\quad [(\mathit{src}.\mathit{id} = \mathit{id}) \wedge \mathit{ack}](\mathit{snd}(\mathit{leader}(\mathit{max})).\mathit{node}(\mathit{id}, F, F, \mathit{max}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num})) \diamond 0\ +\\[1ex]
\quad [(\mathit{parent} \neq ?) \wedge \neg \mathit{ack} \wedge \mathit{elec}]\\
\qquad \mathit{sense}(\mathit{parent}, \mathit{node}(\mathit{id}, \mathit{elec}, \mathit{ack}, \mathit{lid}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num}),\\
\qquad\quad \mathit{snd}(\mathit{leader}(\mathit{max})).\mathit{node}(\mathit{id}, F, F, \mathit{max}, \mathit{max}, \mathit{id}, \mathit{src}, \mathit{num})) \diamond 0\ +\\[1ex]
\quad \sum_{\mathit{li}:\mathit{Loc}} \mathit{rcv}(\mathit{leader}(\mathit{li})).([\neg \mathit{elec} \wedge \mathit{lid} > \mathit{li})]\\
\qquad \mathit{snd}(\mathit{leader}(\mathit{lid})).\mathit{node}(\mathit{id}, \mathit{elec}, \mathit{ack}, \mathit{lid}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num}) \diamond (\\
\qquad\ [(\neg \mathit{ack} \wedge \mathit{elec} \wedge \mathit{li} \geq \mathit{max}) \vee (\neg \mathit{elec} \wedge \mathit{li} > \mathit{lid})]\\
\qquad\quad \mathit{snd}(\mathit{leader}(\mathit{li})).\mathit{node}(\mathit{id}, F, \mathit{ack}, \mathit{li}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num}) \diamond\\
\qquad\quad \mathit{node}(\mathit{id}, \mathit{elec}, \mathit{ack}, \mathit{lid}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num})))\ +\\[1ex]
\quad [(\mathit{lid} \neq ?) \wedge \neg \mathit{elec}]\mathit{finish}(\mathit{lid}, \mathit{id}).\mathit{node}(\mathit{id}, \mathit{elec}, \mathit{ack}, \mathit{lid}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num}) \diamond 0\ +\\[1ex]
\quad [(\mathit{lid} \neq ?) \wedge (\mathit{lid} \neq \mathit{id}) \wedge \neg \mathit{elec}]\\
\qquad \mathit{sense}(\mathit{lid},\ \mathit{snd}(\mathit{leader}(\mathit{lid})).\mathit{node}(\mathit{id}, \mathit{elec}, \mathit{ack}, \mathit{lid}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num}),\\
\qquad\ \ \mathit{node}(\mathit{id}, \mathit{elec}, \mathit{ack}, \mathit{lid}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num}))\ +\\[1ex]
\quad [\mathit{lid} = \mathit{id}]\mathit{snd}(\mathit{leader}(\mathit{lid})).\mathit{node}(\mathit{id}, \mathit{elec}, \mathit{ack}, \mathit{lid}, \mathit{max}, \mathit{parent}, \mathit{src}, \mathit{num}) \diamond 0
\end{array}
$$

Fig. 10 Specification of the leader election algorithm for MANETs [33].

where src identifies the election process in which the node is involved. Then ack is set to false.

Upon receiving an ack message, as specified by the fifth summand, it is processed if the receiving node has not previously sent its ack and the value contained in the ack message (i.e., mid) is greater than the maximum value the node has seen so far. Then the node updates its maximum identifier from max to mid. (Although max is updated upon receipt of an ack message, in [33] a node announces the maximum value its knows by sending the message $\mathit{ack}(\mathit{src}, \mathit{lid})$.)

The sixth summand specifies that a root node announces the node with the largest value it knows as the leader, if it has not previously announced it (i.e., $(\mathit{src}.\mathit{id} = \mathit{id}) \wedge \mathit{ack}$). Then it sets its lid to max and terminates the election process by setting elec and ack to false.

The seventh summand specifies the case that a node previously sent ack and is waiting to receive the $leader$ from its parent (i.e., $(parent \neq ?) \wedge \neg ack \wedge elec$), but gets disconnected from its parent (examined by the sense operator). The node then acts as the root of its subtree by announcing the leader.

The eighth summand specifies how a node handles received $leader$ messages. In case such a message contains an identifier higher than the node's leader and the node's maximum value (after it informed its parent about this maximum value, i.e., $\neg ack \wedge elec$), it adopts that identifier as its leader, and announces the new leader to inform its neighbors. Otherwise it announces its own leader to the sender.

With the aim to model checking certain properties, to observe the leader of a node, a self-loop is added to states with the termination status (i.e., $(lid \neq ?) \wedge \neg elec$). It performs the action *finish*, parameterized with the node's leader and identifier. This is specified by the ninth summand.

The tenth summand specifies that non-leader nodes periodically announce their leader, if they are still connected to their parent (examined by the sense operator), Owing to these messages, if two spanning trees are joined, nodes in one spanning tree are informed about the leader of the other spanning tree. As specified by the eighth summand, the leader with the higher identifier is selected as the leader of the merger of the two spanning trees.

Similarly, leader nodes periodically announce their existence, as specified by the last summand. (In [33], upon receiving a $leader(li)$ message while $\neg elec \wedge lid > lid$, the node wrongly sends $leader(max)$ instead of $leader(lid)$.)

A MANET of four nodes is specified by the parallel composition of $[\![node(\ell, F, F, ?, \ell, ?, dc(0, ?), 0)]\!]_\ell$ for $\ell \in \{A, B, C, D\}$. Furthermore, we close the network on all message types and abstract away communication actions:

$$\tau_{Msg}(\partial_{Msg}([\![node(A, F, F, ?, ?, ?, dc(0, ?), 0)]\!]_A \parallel \ldots \parallel [\![node(D, F, F, ?, ?, ?, dc(0, ?), 0)]\!]_D))$$

7.1.2 Protocol verification

The leader election algorithm for MANETs presented in the previous section requires that *eventually* every node is in a stable state, i.e. has a unique leader which is the highest-valued node in its SCC. We show by means of our model checking algorithm that a MANET with four nodes, each deploying the protocol from [33], has a *weak* form of stabilization: if nodes stay strongly connected, then the algorithm converges to a desired state. Assume four nodes identifiers $A < B < C < D$. We investigate the property that "all nodes eventually choose D as their leader, if they are continuously in the same SCC", specified by the CACTL formula

$$p_1 \equiv \mathbf{AF}^{\mu_1}_{true}(\textstyle\bigwedge_{x \in Loc} \mathbf{EX}_{finish(D,x)} true)$$

where $\mu_1 \equiv A \dashrightarrow B \wedge B \dashrightarrow C \wedge C \dashrightarrow D \wedge D \dashrightarrow A$.

We note that the strong fairness assumption in the CACTL model checking algorithm is essential for **AU** formulae to guarantee that executions will always escape from the self-loops of *finish* actions.

Next we examine merging scenarios of the protocol: when two distinct SCCs merge, nodes adopt the node with the highest value in both as their leader. We can verify the property “if an SCC consisting of A and B is connected to an SCC consisting of C and D, then D becomes the leader of all nodes” with the help of the CACTL formula

$$\begin{aligned} p_2 \;\equiv\;& \mathbf{AG}^{\mathit{true}}_{\mathit{true}}((\textstyle\bigwedge_{x\in\{A,B\}} \mathbf{EX}_{\mathit{finish}(B,x)}\, \mathit{true}) \wedge (\textstyle\bigwedge_{y\in\{C,D\}} \mathbf{EX}_{\mathit{finish}(D,y)}\, \mathit{true}) \\ & \qquad \Rightarrow \mathbf{AF}^{\mu_1}_{\mathit{true}}(\textstyle\bigwedge_{z\in Loc} \mathbf{EX}_{\mathit{finish}(D,z)}\, \mathit{true})) \end{aligned}$$

The formula expresses that first nodes A, B and nodes C, D belong to the same SCC, with leader B and D, respectively. If these two SCCs merge, then they will eventually converge to the same leader D. The property can become more complex by examining a scenario in which three SCCs merge. We can verify the property “if an SCC consisting of A and C is first connected to D and later to B, then D becomes the leader of all nodes” with the help of the CACTL formula

$$\begin{aligned} p_3 \;\equiv\;& \mathbf{AG}^{\mathit{true}}_{\mathit{true}}((\textstyle\bigwedge_{x\in\{A,C\}} \mathbf{EX}_{\mathit{finish}(C,x)}\, \mathit{true}) \wedge (\textstyle\bigwedge_{y\in\{B,D\}} \mathbf{EX}_{\mathit{finish}(y,y)}\, \mathit{true}) \\ & \Rightarrow \mathbf{AF}^{\mu_2}_{\mathit{true}}((\textstyle\bigwedge_{z\in\{A,C,D\}} \mathbf{EX}_{\mathit{finish}(D,z)}\, \mathit{true}) \\ & \qquad \Rightarrow \mathbf{AF}^{\mu_1}_{\mathit{true}}(\textstyle\bigwedge_{w\in Loc} \mathbf{EX}_{\mathit{finish}(D,w)}\, \mathit{true}))) \end{aligned}$$

where $\mu_2 \equiv A \dashrightarrow C \wedge C \dashrightarrow D \wedge D \dashrightarrow A$.

We can investigate scenarios in which a node gets disconnected from its parent or leader during or after the leader election phase respectively. One can verify that “a node will not change its leader as long as it is connected to it”, specified by the CACTL formula

$$p_4 \;\equiv\; \mathbf{AF}^{A \dashrightarrow D \,\wedge\, D \dashrightarrow A}_{\mathit{finish}(D,A)}(\mathbf{AG}^{D \dashrightarrow A}_{\neg(\mathit{leader}(A,A)\vee \mathit{leader}(B,A)\vee \mathit{leader}(C,A))}\, \mathit{true})$$

meaning that after D was selected as the leader of A when $A \dashrightarrow D \wedge D \dashrightarrow A$ (specified by the outer $\mathbf{AF}$ formula), A never chooses another leader (A, B or C), unless it gets disconnected from D.

7.1.3 Tool support

We have implemented the model checking algorithms explained in Section 6 in Java.[3] We exploited the mCRL2 toolset [21] to convert *RRBPT* specifications into LTSs, where actions are parameterized by network constraints. Our model checker tool converts the given LTS to the desired CLTS before any property investigation. The framework of mCRL2 integrates process and abstract data specifications. Network constraints and calculations over them can thus be specified as data terms and functions in the same way as [18].

The main differences between mCRL2 and *RRBPT* are on the deployment and sensing operators of *RRBPT* and their parallel composition. For an *RRBPT* process

[3] The source code is available at `http://rebeca.cs.ru.is/files/Tools/CLTS.zip`.

term t deployed at address ℓ, assume $encode(t, \ell, \mathcal{C})$ denotes the corresponding code in mCRL2, defined inductively by:

$$\begin{aligned}
encode(t_1 + t_2, \ell, \mathcal{C}) &= encode(t_1, \ell, \mathcal{C}) + encode(t_2, \ell, \mathcal{C})\\
encode([b]t_1 \diamond t_2, \ell, \mathcal{C}) &= encode(t_1, \ell, \mathcal{C}) \lhd b \rhd encode(t_2, \ell, \mathcal{C})\\
encode(0, \ell, \mathcal{C}) &= 0
\end{aligned}$$

where $t_1 \lhd b \rhd t_2$ behaves as t_1 if b evaluates to true, and otherwise as t_2. We postpone the explanation of the parameter $\mathcal{C}$ until the explanation of the encoding of the action prefix and $sense$ operator.

To model pairs of network constraints and actions in the semantics of ***RRBPT***, we parameterize network send and receive actions in mCRL2 with network constraints as follows. In mCRL2, two actions α and β are synchronized if their synchronization is defined, denoted by $\alpha \mid \beta = \gamma$, and they agree on the number and values of their parameters. By contrast, in ***RRBPT*** two actions are synchronized if they are either two $nrcv$ actions or an $nrcv$ and an $nsnd$ action, and they agree on the message part, while some calculations are performed on their network constraints (see rules $Sync_{1,2}$). To model the effect of rules Snd and Rcv_{1-3}, and take care of computations over network constraints defined by rules $Sync_{1,2}$, we define a set of actions $nsnd_{ij}, nrcv_{ij} : \mathbb{C} \times Msg \times Loc$, where $0 \leq i \leq n$ and $1 \leq j \leq n$ with n the number of nodes. For network addresses Loc, assume L_c, L_d and ℓ such that $Loc = L_c \cup L_d \cup \{\ell\}$ and $L_c \cap L_d = \emptyset$ and $\ell \notin L_c \cup L_d$. Furthermore, let $\mathcal{C}(L_c, L_d, \ell)$ denote the network constraint $\{\ell \rightsquigarrow \ell' \mid \ell' \in L_c\} \cup \{\ell \not\rightsquigarrow \ell' \mid \ell' \in L_d\}$, and $|S|$ the size of set S. The action $nrcv_{ij}(\mathcal{C}(L_c, L_d, \ell), \mathfrak{m}, \ell)$, where $|L_c| = j$ and $i \leq j$, denotes that when the message $\mathfrak{m}$ is sent by the node with address ℓ, i nodes with addresses in L_c are ready to receive it because they are connected to the sender, while nodes with addresses in L_d cannot receive it as they are disconnected from the sender. And $nsnd_{i,j}(\mathcal{C}(L_c, L_d, \ell), \mathfrak{m}, \ell)$, where $|L_c| = j$ and $i \leq j$, denotes that the node with address ℓ has sent the message $\mathfrak{m}$, while i nodes with addresses in L_c have received it, but nodes with addresses in L_d cannot receive it as they are disconnected from the sender. The allowed synchronizations are $nrcv_{ij} \mid nrcv_{kj} = nrcv_{(i+k)j}$, where $i + k \leq j$, and $nrcv_{ij} \mid nsnd_{kj} = nsnd_{(i+k)j}$, where $i + k \leq j$.

In encoding ***RRBPT*** processes in mCRL2, deployment operators are removed, and protocol send and receive actions are modeled by calling $sndProcess(\mathfrak{m}, \ell, \mathcal{C})$ and $rcvProcess(\mathfrak{m}, \ell, \mathcal{C})$ respectively, which are defined below. Here ℓ is the address of deployment and network constraint $\mathcal{C}$ defines the status of the links in the network; as a result the network behavior is restricted accordingly. This latter parameter is set to empty to allow any possible behavior of the network. The processes unfold send and receive actions regarding how the node that runs the actions can be synchronized in the network.

$$
\begin{aligned}
&\mathit{encode}(\mathit{rcv}(m).t, \ell, \mathcal{C}) = \mathit{rcvProcess}(\mathfrak{m}, \ell, \mathcal{C}).\mathit{encode}(t, \ell, \{\})\\
&\mathit{encode}(\mathit{snd}(m).t, \ell, \mathcal{C}) = \mathit{sndProcess}(\mathfrak{m}, \ell, \mathcal{C}).\mathit{encode}(t, \ell, \{\})\\
&\mathit{rcvProcess}(\mathfrak{m}, \ell, \mathcal{C}') = \textstyle\sum_{\ell' \in Loc}[\ell = \ell']\\
&\quad (\textstyle\sum_{L_c \subseteq Loc(\ell' \in L_c)} \sum_{L_d \subseteq Loc} \sum_{\ell'' \in Loc(\ell'' \neq \ell')} \sum_{j>0} \sum_{i \leq j}\\
&\qquad [\neg \mathcal{C}' \cap \mathcal{C}(L_c, L_d, \ell'') = \emptyset] \mathit{nrcv}_{ij}(\mathcal{C}(L_c, L_d, \ell''), \mathfrak{m}, \ell'')) \diamond 0\\
&\mathit{sndProcess}(\mathfrak{m}, \ell, \mathcal{C}') = \textstyle\sum_{\ell' \in Loc}[\ell = \ell']\\
&\quad (\textstyle\sum_{L_c \subseteq Loc} \sum_{L_d \subseteq Loc} \sum_{j \geq 0} \sum_{i \leq j} [\neg \mathcal{C}' \cap \mathcal{C}(L_c, L_d, \ell') = \emptyset]\\
&\qquad \mathit{nsnd}_{ij}(\mathcal{C}(L_c, L_d, \ell'), \mathfrak{m}, \ell')) \diamond 0
\end{aligned}
$$

To model the process term $\mathit{sense}(\ell', t_1, t_2)$ deployed at the node with address ℓ, a sense action is performed, parameterized by a network constraint indicating the link status. Furthermore, the appropriate link status is added to the network constraint to encode t_1 and t_2:

$$
\begin{aligned}
&\mathit{encode}(\mathit{sense}(\ell', t_1, t_2), \ell, \mathcal{C}) = \mathit{sense}(\{\ell' \rightsquigarrow \ell\}).\mathit{encode}(t_1, \ell, \mathcal{C} \cup \{\ell' \rightsquigarrow \ell\}) +\\
&\quad \mathit{sense}(\{\ell' \not\rightsquigarrow \ell\}).\mathit{encode}(t_2, \ell, \mathcal{C} \cup \{\ell' \not\rightsquigarrow \ell\})
\end{aligned}
$$

The accumulated network constraints restrict communication of the network as explained in encoding the receive actions. The sense action of a node is synchronized with sensed actions of other nodes to ensure that all nodes adhere to the same restrictions on the network. To this aim, a set of sensed_1 actions is added to each process name definition. For instance, the process definition $X(d : D) \overset{def}{=} t$ is encoded as:

$$
\begin{aligned}
X(d : D, x : Loc, c : \mathbb{C}) = {} & \mathit{encode}(t, x, c) +\\
& \textstyle\sum_{\ell, \ell' \in Loc} \mathit{sensed}_1(\{\ell \rightsquigarrow \ell'\}).X(d, x, \{\ell \rightsquigarrow \ell'\}) +\\
& \qquad \mathit{sensed}_1(\{\ell \not\rightsquigarrow \ell'\}).X(d, x, \{\ell \not\rightsquigarrow \ell'\})
\end{aligned}
$$

with $\mathit{sensed}_i \mid \mathit{sense} = \mathit{csense}_i$, $\mathit{sensed}_i \mid \mathit{sensed}_j = \mathit{sensed}_{i+j}$, and $\mathit{csense}_i \mid \mathit{sensed}_j = \mathit{csense}_{i+j}$. Consequently, $\mathit{encode}(X(u), \ell, \mathcal{C}) = X(u, \ell, \mathcal{C})$.

The *RRBPT* term $\partial_{Msg}([\![t_1]\!]_{\ell_1} \parallel \ldots \parallel [\![t_n]\!]_{\ell_n})$, where Msg is the set of all messages, is modeled by the mCRL2 operators renaming ρ, encapsulation ∂, and parallel $\parallel$, where $\rho_{\{a \to b\}}$ renames the action name a to b, and $\partial_{\{a\}}$ renames action a to deadlock:

$$
\begin{aligned}
&\rho_{\{\forall i \leq n (\mathit{nsnd}_{ii} \to \mathit{nsnd}, \mathit{csense}_i \to \mathit{sense})\}}(\partial_{\forall i, j \leq n, i \neq j (\{\mathit{nsnd}_{ij}, \mathit{nrcv}_{ij}, \mathit{csense}_i, \mathit{sensed}_i, \mathit{sense}_i)\}}\\
&\qquad ((\mathit{encode}(t_1, \ell_1, \{\}) \parallel \ldots \parallel \mathit{encode}(t_n, \ell_n, \{\})))).
\end{aligned}
$$

The LTS resulting from this encoding contains labels of the form $\mathit{nsnd}(\mathcal{C}, \mathfrak{m}, \ell)$, $\mathit{sense}(\mathcal{C})$, and internal actions.

We encoded the leader election case study from Section 7.1.[4] Our model checker loads the output of mCRL2 by converting all labels of $\mathit{nsnd}(\mathcal{C}, \mathfrak{m}, \ell)$ to $(\mathcal{C}, \tau)$ to

[4] This encoding is available at `https://www.dropbox.com/s/grg0u3s3kroabs6/leaderelection.zip?dl=0`.

model the abstraction operator. For internal actions such as $\mathit{finish}(\ell, \ell', \mathcal{C})$, the label $(\mathcal{C}, \mathit{finish}(\ell, \ell'))$ is generated.

We verified properties p_{1-4}; these properties are all satisfied by our model. We used the mCRL2 tool to generate the state space, modulo strong bisimilarity. Table 3 illustrates the verification time on a Corei7 CPU with 8GB RAM.

Table 3 Verification times of the CACTL model checker.

no. nodes	no. states	no. transitions	p_1	p_2	p_3	p_4
3	$4,278$	$33,536$	0.303s	0.358s	0.369s	0.275s
4	$357,024$	$3,928,890$	36.554s	53.204s	62.763s	91.190s

Before reduction modulo strong bisimilarity, our approach yields state spaces with $37,992$ states and $381,912$ transitions and $11,922,272$ states and $150,908,802$ transitions for the network of three and four nodes, respectively. By contrast, the classical approach yields state spaces that are 8 and 64 times larger in the number of states for the network of three and four nodes, respectively. Our modeling approach reduces the state space substantially, and our model checking algorithm performs significantly better than existing model checkers for MANET protocols. Furthermore, our approach is flexible in defining topology constraints to investigate different mobility scenarios.

8 Conclusion and Future Work

We presented a branching-time temporal logic CACTL, which is interpreted over CLTSs. It is aimed at expressing topology-dependent behavioral properties of MANET protocols. We moreover introduced a model checking algorithm for CACTL that investigates the temporal behavior of MANETs, while taking into account the underlying topology, represented symbolically by means of network constraints. Scenarios like *after a route found* and *after two distinct SCCs are merged* can be investigated with the help of multi-hop constraints over topologies, which in CACTL are specified on top of path quantifiers.

Advantages of our approach are flexibility in verifying topology-dependent behavior (without changing the model), and restricting the generality of mobility. By nesting path quantifiers, a set of specific topological paths can be specified with the help of topology constraints (without a need to specify how a topology constraint should be inferred). The (dis)connectivity information in CLTS transitions makes it possible to restrict the generality of mobility as desired.

We intend to extend our model checker by generating counter-examples if a property is not satisfied. Furthermore, we intend to improve the expressiveness of our logic by extending topology formulae with negative multi-hop conditions to support properties depending on non-existence of a multi-hop connection. To this aim, the definition of an invalid path for a multi-hop network constraint should be adopted to prevent links from violating a negative multi-hop condition. We expect that such an adoption can be implemented efficiently in the model checker.

Acknowledgements

Jan Friso Groote provided vital support in our employment of the mCRL2 toolset. Saeide Ahmadi helped with the implementation of an earlier CACTL model checker, in Maude.

References

1. Agha, G.A.: ACTORS - A Model of Concurrent Computation in Distributed Systems. MIT Press (1990)
2. wRebeca, Efficient Modeling of Mobile Ad hoc Networks. `http://fghassemi.adhoc.ir/wrebeca`
3. Bhargavan, K., Obradovic, D., Gunter, C.: Formal verification of standards for distance vector routing protocols. Journal of the ACM **49**(4), 538–576 (2002)
4. Blom, S., Fokkink, W., Groote, J.F., van Langevelde, I., Lisser, B., van de Pol, J.: μCRL: A toolset for analysing algebraic specifications. In: Proc. 13th Conference on Computer Aided Verification, *Lecture Notes in Computer Science*, vol. 2102, pp. 250–254. Springer (2001)
5. Bourke, T., van Glabbeek, R., Höfner, P.: A mechanized proof of loop freedom of the (untimed) AODV routing protocol. In: Proc. 12th Symposium on Automated Technology for Verification and Analysis, *Lecture Notes in Computer Science*, vol. 8837, pp. 47–63. Springer (2014)
6. Chang, E.J.H.: Echo algorithms: Depth parallel operations on general graphs. IEEE Transactions on Software Engineering **8**(4), 391–401 (1982)
7. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Logic of Programs, *Lecture Notes in Computer Science*, vol. 131, pp. 52–71. Springer (1981)
8. Clarke, E.M., Emerson, E.A., Sistla, A.P.: automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transaction of Programming Languages and Systems **8**(2), 244–263 (1986)
9. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press (2001)
10. De Nicola, R., Fantechi, A., Gnesi, S., Ristori, G.: An action-based framework for verifying logical and behavioural properties of concurrent systems. Computer Networks and ISDN Systems **25**(7), 761–778 (1993)
11. De Nicola, R., Vaandrager, F.: Action versus state based logics for transition systems. In: Semantics of Systems of Concurrent Processes, *Lecture Notes in Computer Science*, vol. 469, pp. 407–419. Springer (1990)
12. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.: Automated analysis of AODV using UPPAAL. In: Proc. 18th Conference on Tools and Algorithms for the Construction and Analysis of Systems, *Lecture Notes in Computer Science*, vol. 7214, pp. 173–187. Springer (2012)
13. Fehnker, A., van Glabbeek, R., Höfner, P., McIver, A., Portmann, M., Tan, W.: A process algebra for wireless mesh networks. In: Proc. 21st European Symposium on Programming, *Lecture Notes in Computer Science*, vol. 7211, pp. 295–315. Springer (2012)
14. Fokkink, W.: Modelling Distributed Systems. Springer (2007)
15. Fokkink, W.: Distributed Algorithms: An Intuitive Approach. MIT Press (2013)
16. Ghassemi, F., Ahmadi, S., Fokkink, W., Movaghar, A.: Model checking MANETs with arbitrary mobility. In: Proc. 5th Conference on Fundamentals of Software Engineering, *Lecture Notes in Computer Science*, vol. 8161, pp. 217–232. Springer (2013)
17. Ghassemi, F., Fokkink, W., Movaghar, A.: Equational reasoning on mobile ad hoc networks. Fundamenta Informaticae **103**(1), 1–41 (2010)
18. Ghassemi, F., Fokkink, W., Movaghar, A.: Verification of mobile ad hoc networks: An algebraic approach. Theoretical Computer Science **412**(28), 3262–3282 (2011)
19. Ghassemi, F., Talebi, M., Movaghar, A., Fokkink, W.: Stochastic restricted broadcast process theory. In: Proc. 8th European Performance Engineering Workshop, *Lecture Notes in Computer Science*, vol. 6977, pp. 72–86. Springer (2011)
20. Godskesen, J.: A calculus for mobile ad hoc networks. In: Proc. 9th Conference on Coordination Models and Languages, *Lecture Notes in Computer Science*, vol. 4467, pp. 132–150. Springer (2007)

21. Groote, J.F., Mousavi, M.R.: Modeling and Analysis of Communicating Systems. MIT Press (2014)
22. Hammer, M., Weber, M.: "To store or not to store" reloaded: Reclaiming memory on demand. In: Proc. 11th Workshop on Formal Methods for Industrial Critical Systems, *Lecture Notes in Computer Science*, vol. 4346, pp. 51–66. Springer (2006)
23. Kouzapas, D., Philippou, A.: A process calculus for dynamic networks. In: Formal Techniques for Distributed Systems, *Lecture Notes in Computer Science*, vol. 6722, pp. 213–227. Springer (2011)
24. McIver, A., Fehnker, A.: Formal techniques for analysis of wireless networks. In: Proc. 2nd Symposium on Leveraging Applications of Formal Methods, pp. 263–270. IEEE (2006)
25. Meolic, R., Kapus, T., Brezocnik, Z.: ACTLW - An action-based computation tree logic with unless operator. Information Sciences **178**(6), 1542–1557 (2008)
26. Merro, M.: An observational theory for mobile ad hoc networks. In: Proc. 23rd Conference on the Mathematical Foundations of Programming Semantics, *Electronic Notes in Theoretical Computer Science*, vol. 173, pp. 275–293. Elsevier (2007)
27. Mezzetti, N., Sangiorgi, D.: Towards a calculus for wireless systems. In: Proc. 22nd Conference on Mathematical Foundations of Programming Semantics, *Electronic Notes in Theoretical Computer Science*, vol. 158, pp. 331–353. Elsevier (2006)
28. Nanz, S., Hankin, C.: A framework for security analysis of mobile wireless networks. Theoretical Computer Science **367**(1), 203–227 (2006)
29. Nanz, S., Nielson, F., Nielson, H.: Static analysis of topology-dependent broadcast networks. Information and Computation **208**(2), 117–139 (2010)
30. de Renesse, R., Aghvami, A.: Formal verification of ad-hoc routing protocols using SPIN model checker. In: Proc. 12th Mediterranean Electrotechnical Conference, pp. 1177–1182. IEEE (2004)
31. Singh, A., Ramakrishnan, C.R., Smolka, S.A.: A process calculus for mobile ad hoc networks. Science of Computer Programming **75**(6), 440–469 (2010)
32. Sirjani, M., Movaghar, A., Shali, A., de Boer, F.: Modeling and verification of reactive systems using Rebeca. Fundamenta Informaticae **63**(4), 385–410 (2004)
33. Vasudevan, S., Kurose, J., Towsley, D.: Design and analysis of a leader election algorithm for mobile ad hoc networks. In: 12th Conference on Network Protocols, pp. 350–360. IEEE (2004)
34. Wibling, O., Parrow, J., Pears, A.: Automatized verification of ad hoc routing protocols. In: Proc. 24th IFIP Conference on Formal Techniques for Networked and Distributed Systems, *Lecture Notes in Computer Science*, vol. 3235, pp. 343–358. Springer (2004)
35. Wibling, O., Parrow, J., Pears, A.: Ad hoc routing protocol verification through broadcast abstraction. In: Proc. 25th IFIP Conference on Formal Techniques for Networked and Distributed Systems, *Lecture Notes in Computer Science*, vol. 3731, pp. 128–142. Springer (2005)
36. Yousefi, B., Ghassemi, F., Khosravi, R.: Modeling and efficient verification of wireless ad hoc network. `http://arxiv.org/abs/1604.07179`