Theoretical Computer Science 412 (2011) 3262–3282

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

# Verification of mobile ad hoc networks: An algebraic approach

Fatemeh Ghassemi [a], Wan Fokkink [b,*], Ali Movaghar [a]

[a] *Sharif University of Technology, Tehran, Iran*

[b] *Vrije Universiteit, Amsterdam, The Netherlands*

A R T I C L E I N F O

*Keywords:*
Process theory
Local broadcast
Mobility
Axiomatization
Uniform MANET
Symbolic verification

A B S T R A C T

We introduced Computed Network Process Theory to reason about protocols for mobile ad hoc networks (MANETs). Here we explore the applicability of our framework in two regards: model checking and equational reasoning. The operational semantics of our framework is based on constrained labeled transition systems (*CLTS*s), in which each transition label is parameterized with the set of topologies for which this transition is enabled. We illustrate how through model checking on *CLTS*s one can analyze mobility scenarios of MANET protocols. Furthermore, we show how by equational theory one can reason about MANETs consisting of a finite but unbounded set of nodes, in which all nodes deploy the same protocol. Model checking and equational reasoning together provide us with an appropriate framework to prove the correctness of MANETs. We demonstrate the applicability of our framework by a case study on a simple routing protocol.



## 1. Introduction

Mobile ad hoc networks (MANETs) consist of mobile nodes equipped with wireless transceivers to communicate with each other directly or along multihop paths. Two nodes can effectively communicate if they are located in the communication range of each other, defined by the underlying topology. Wireless communication is inherently unreliable; a node may not succeed in communicating due to noise in the environment. Moreover, the mobility of nodes makes the underlying topology dynamic. The characteristics of wireless communication and dynamism of the underlying topology require a suitable framework for the modeling and verification of MANETs.

We introduced *Restricted Broadcast Process Theory* (*RBPT*) in [1] to specify and verify MANET protocols, taking into account the mobility of nodes. Topology changes are modeled implicitly in the semantics, and thus one can verify a network with respect to arbitrary topology changes. *Computed Network Process Theory* (*CNT*) [2,3] is an extension of *RBPT* with so-called computed network terms and auxiliary operators, as an expedient verification framework with a sound and complete axiom system, modulo so-called rooted branching computed network bisimilarity. The operational semantics of *CNT* is given by constrained labeled transition systems (*CLTS*s), in which each transition label is parameterized by a set of topologies for which this transition is enabled.

In this paper, we enhance and illustrate the applicability of our framework for the verification of MANETs in two regards: model checking and equational reasoning. We show how the semantic model of *CLTS*s makes it possible to derive and analyze mobility scenarios for MANET protocols. We exploit the mCRL2 toolset [4] to convert *CNT* specifications into *CLTS*s, and then the CADP toolset [5] to verify properties.

To verify MANET protocols for large networks, or if one needs to deal with infinite data domains, model checking is not readily applicable. Since MANETs often consist of an arbitrary set of nodes that run the same protocols, we develop a

* Corresponding author. Tel.: +31 20 5987735; fax: +31 20 5987653.
*E-mail addresses:* fghassemi@mehr.sharif.edu (F. Ghassemi), w.j.fokkink@vu.nl, wanf@cs.vu.nl (W. Fokkink), movaghar@sharif.edu (A. Movaghar).

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.03.017

symbolic verification technique for such networks within the *CNT* framework, based on the cones and foci method [6–8]. This technique works on a restricted class of specifications, called linear computed network equations, in which the states are data objects, and rephrases the question whether the system specification and implementation are equivalent in terms of proof obligations on relations between data objects. We exploit our equations to convert the parallel composition of an arbitrary number of similar processes, modulo some data parameters, to a single linear equation using the Composition Theorem from [9]. The Composition Theorem however is based on the assumption that communications are restricted to two processes. Since in our framework wireless communication is an essential ingredient, we generalize the Composition Theorem to this setting. The linear equation representing the MANET of similar nodes, and the desired external behavior of this network (also expressed by a linear equation) are taken as input to the symbolic verification technique, which reduces the question of their behavioral equivalence to proving data equalities. The framework of mCRL2 [10] allows for this tight integration between processes and data.

To the best of our knowledge, this paper is the first to address the symbolic verification of MANETs. We use a simple routing protocol based on ad hoc on-demand distance vector (AODV) routing protocol as a running example, and prove that the protocol correctly routes data from a source to a destination.

The structure of the paper is as follows. Section 2 explains the modeling concepts and operational semantics underlying *CNT*. Section 3 explains the formal framework: syntax and axioms. Section 4 presents a case study, and shows how our semantic model is capable of deriving errors caused by the mobility of nodes. In Section 5 we explain our symbolic verification approach, and how it can be exploited to verify MANETs with similar nodes. Finally, Section 6 summarizes our results and future work.

## 2. Concepts

In wireless communication, when a node transmits a message, only nodes that are located in its transmission area can receive this message. For this reason, the communication in wireless networks is called local broadcast. We model the unreliable local broadcast service provided by the MAC-layer (of each MANET node), in which the sender and receiver are synchronous and receive actions carry (error-free) messages. A node $B$ is *connected to* a node $A$, if $B$ is located within the transmission range of $A$. This connectivity relation between nodes, which is not necessarily symmetric, introduces a *topology* concept. A topology is a function $\gamma : Loc \rightarrow \mathbb{P}(Loc)$ where $A, B, C \in Loc$ denote a finite set of addresses, which models the hardware addresses. The network topology, due to the mobility of nodes in a MANET, is dynamic and may change rapidly and unpredictably over time.

We model mobility implicitly in the semantics; each state is representative of all possible topologies a network can meet, and a network can be at any of these topologies. Each transition is constrained by a set of topologies for which such a behavior is possible. We introduced *network constraints* in [2] to formally specify the set of topologies. The set of addresses is extended with the unknown address ?. A network constraint $\mathcal{C}$ is a set of connectivity pairs $\rightsquigarrow$: $Loc \times Loc$, such that the second address cannot be ?. The connectivity pair $? \rightsquigarrow A$ denotes that a node with address $A$ is connected to an unknown address from which it can receive data, while $B \rightsquigarrow A$ denotes that $A$ is connected to $B$ and consequently $B$ can send data to $A$. We write $\{B \rightsquigarrow A, C\}$ instead of $\{B \rightsquigarrow A, B \rightsquigarrow C\}$. Each network constraint $\mathcal{C}$ represents the set of topologies that satisfy the connectivity pairs in $\mathcal{C}$, i.e., $\{\gamma | \forall \ell \in Loc \cdot \mathcal{C}(\ell) \subseteq \gamma(\ell)\}$. Therefore, the empty network constraint {} denotes all possible topologies. Let $\mathbb{C}$ denote the set of all network constraints that can be defined over network addresses in $Loc$.

In this paper, compared to [3], we transfer network constraints from transition subscripts into transition labels, which are interpreted as the set of topologies for which such a transition is enabled. This transmission of network constraints from the transitions into the labels simplifies our framework, as will be explained in Section 3.2. We override the definition of *constrained labeled transition systems* (*CLTS*s), the operational behavior of MANETs, given in [3] such that the previous results are preserved.

Let $Msg$ denote a set of messages communicated over a network and ranged over by $\mathtt{m}$. Let $Act$ be the network send and receive actions with signatures $nsnd : Msg \times Loc$ and $nrcv : Msg$ respectively. The send action $nsnd(\mathtt{m}, \ell)$ denotes that the message $\mathtt{m}$ is transmitted from a node with the address $\ell$, while the receive action $nrcv(\mathtt{m})$ denotes that the message $\mathtt{m}$ is ready to be received. Let $Act_\tau = Act \cup \{\tau\}$, ranged over by $\eta$.

**Definition 1.** A *CLTS* is defined by $\langle S, L, \rightarrow, s_0 \rangle$, with $S$ a set of states, $L \subseteq \mathbb{C} \times Act_\tau$, $\rightarrow \subseteq S \times L \times S$ a transition relation, and $s_0 \in s$ the initial state. A transition $(s, (\mathcal{C}, \eta), s') \in \rightarrow$ is denoted by $s \xrightarrow{(\mathcal{C},\eta)} s'$.

Suppose for a real MANET (a network with a set of nodes, each running a process), its behavior is modeled by a *CLTS* partly shown in Fig. 1. Consider the behavior of the real MANET when all processes of nodes are reset. We explain how the behavior of this MANET for the scenario depicted in Fig. 2 can be inferred from its *CLTS* model. Initially its *CLTS* is in state $s_0$ irrespective of the underlying topology. When the underlying topology changes from 2.1 to 2.2 in the MANET, the state of its *CLTS* is not changed, since each state is representative of any topology changes. When the underlying topology is 2.2, the message $\mathtt{m}_1$ is sent by node $A$. Since the underlying topology belongs to the network constraints {} and $\{A \rightsquigarrow B\}$, the *CLTS* implies that the behavior of MANET would become state $s_0$ or $s_1$. Being in any of these states, when the underlying topology of MANET changes to 2.3 and then 2.4, the state of its model is not changed. However, when the underlying topology is 2.4, node $B$ sends $\mathtt{m}_2$. Since a transition with such an action does not belong to state $s_0$, it can be inferred that the previous

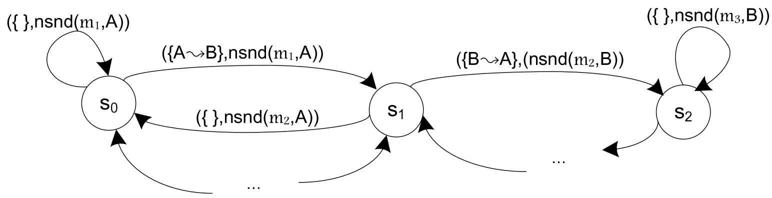


**Fig. 1.** A part of the behavioral model of a MANET.

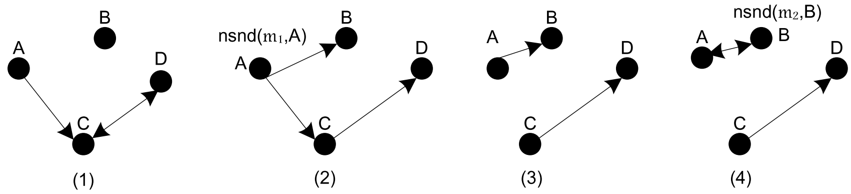


**Fig. 2.** A mobility scenario.

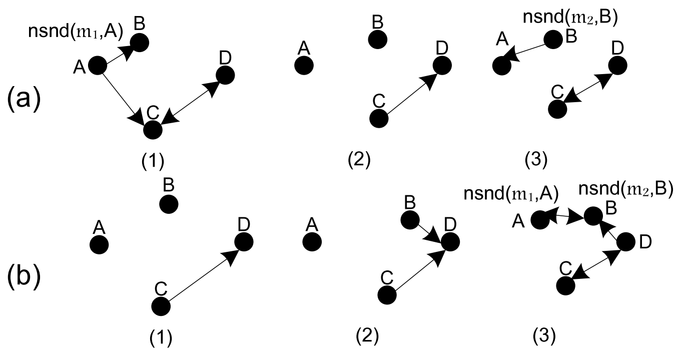


**Fig. 3.** Two mobility scenarios for an execution fragment of the *CLTS* in Fig. 1.

behavior of MANET was $s_1$. The model explains that since the underlying topology belongs to the network constraints {} and $\{B \rightsquigarrow A\}$, the next behavior of the MANET would be either $s_0$ or $s_2$.

For the executions of the *CLTS* in Fig. 1, such as $s_0 \xrightarrow{(\{A\rightsquigarrow B\},nsnd(\mathfrak{m}_1,A))} s_1 \xrightarrow{(\{B\rightsquigarrow A\},nsnd(\mathfrak{m}_2,B))} s_2$, we can derive different mobility scenarios for the real MANET, as shown in Fig. 3.

Concluding, a *CLTS* defines the behavior of the corresponding MANET for arbitrary topology changes, and an execution of the *CLTS* represents multiple mobility scenarios.

## 3. Formal framework: computed network theory

To separate the manipulation of data from processes, we make use of equational abstract data types [11]. Data is specified by equational specifications: one can declare data types (so-called *sorts*) and functions working upon these data types, and describe the meaning of these functions by equational axioms. Following the approach of [12,13], we consider the Computed Network Theory with equational abstract data types. We first explain the set of data types considered in our framework, and then define the *CNT* operators and their axioms.

### 3.1. Data types

We treat the set of network addresses *Loc*, messages *Msg* and network constraints $\mathbb{C}$ as data types within the *CNT* framework. By defining appropriate functions over them, we can provide the axioms and operational semantics of computed network terms. We use mCRL2 notation to define data types: *sort* declares sort names, *func* specifies constructor and *map* non-constructor functions, *var* declares variable names, and *rew* defines non-constructor functions by means of rewrite rules. We assume that the function $if : Bool \times D \times D$ is defined for all data sorts $D$, which returns the first $D$ parameter if the boolean parameter equals true, otherwise the second $D$ parameter is returned.

The data sort *Bool* is used in the conditional operator construct to change the behavior of a process in terms of data values. This data sort is defined by two constructors $T$ and $F$. The conventional operators $\wedge$, $\vee$ and $\neg$ can be defined over it

$$
\begin{array}{llll}
\mathit{sort} & \mathit{Msg} & \mathit{sort} & \mathit{Loc} \\
\mathit{func} & \mathit{req} : \mathit{Loc} \to \mathit{Msg} & \mathit{func} & ? :\to \mathit{Loc} \\
 & \mathit{rep} : \mathit{Loc} \times \mathit{Loc} \to \mathit{Msg} & & \mathit{adr} : \mathit{Loc} \to \mathit{Loc} \\
\mathit{map} & \mathit{isType}_{\mathit{req}} : \mathit{Msg} \to \mathit{Bool} & \mathit{map} & \mathit{eq} : \mathit{Loc} \times \mathit{Loc} \to \mathit{Bool} \\
 & \mathit{eq} : \mathit{Msg} \times \mathit{Msg} \to \mathit{Bool} & & >: \mathit{Loc} \times \mathit{Loc} \to \mathit{Bool} \\
\mathit{var} & \ell, \ell_1, \ell_2, \ell_3, \ell_4 : \mathit{Loc} & & \\
\mathit{rew} & \mathit{eq}(\mathit{req}(\ell_1), \mathit{req}(\ell_2)) = \mathit{eq}(\ell_1, \ell_2) & \mathit{sort} & \mathbb{C} \\
 & \mathit{eq}(\mathit{rep}(\ell_1, \ell_2), \mathit{rep}(\ell_3, \ell_4)) = & \mathit{func} & \mathit{empNC} :\to \mathbb{C} \\
 & \quad \mathit{eq}(\ell_1, \ell_3) \wedge \mathit{eq}(\ell_2, \ell_4) & & \mathit{con} : \mathit{Loc} \times \mathit{Loc} \times \mathbb{C} \to \mathbb{C} \\
 & \mathit{isType}_{\mathit{req}}(\mathit{req}(\ell)) = T & \mathit{map} & \mathit{union} : \mathbb{C} \times \mathbb{C} \to \mathbb{C} \\
 & \mathit{isType}_{\mathit{req}}(\mathit{rep}(\ell_1, \ell_2)) = F & & \mathit{subs} : \mathit{Loc} \times \mathit{Loc} \times \mathbb{C} \to \mathbb{C} \\
 & & & \mathit{include} : \mathbb{C} \times \mathbb{C} \to \mathit{Bool}
\end{array}
$$

**Fig. 4.** Data sorts used in the *CNT* framework.

straightforwardly. The data sort *Nat* specifies the natural numbers by the constant 0 and the unary function *succ*. We use $1, 2, \ldots$ for $succ(0), succ(succ(0)), \ldots$. The definition of functions $+, >, \geq$ and *eq* are straightforward.

Some data sort definitions are given in Fig. 4. For a complete definition see [14]. The network addresses are generated from the constant ? and the unary function *adr*. We use $A, B, \ldots$ to denote $adr(?), adr(adr(?)), \ldots$. The functions *eq* and $>$ compare two network addresses. The network constraints are generated from the constant *empNC* and the *con* function which adds a connectivity pair to network constraints. The function *union* merges two network constraints such that the redundant connections are removed and the connectivity pairs are sorted in terms of the connected addresses (i.e. the second parameter in *con*). The function *subs* substitutes all occurrences of the address in its second parameter with the address in its first parameter. The function *include* examines if the connectivity pairs of a network constraint are included in another. We write $\mathcal{C}_1 \cup \mathcal{C}_2$, $\mathcal{C}_1 \subseteq \mathcal{C}_2$ and $\mathcal{C}[\ell/\ell']$ instead of $union(\mathcal{C}_1, \mathcal{C}_2)$, $include(\mathcal{C}_1, \mathcal{C}_2)$ and $subs(\mathcal{C}, \ell, \ell')$ respectively. We also write $\{\}$, $\{A \rightsquigarrow B\}$, $\{A \rightsquigarrow B, C\}$ for $empNC$, $con(A, B, empNC)$, $con(A, B, con(A, C, empNC))$.

A message can carry data parameters. For instance, in Fig. 4, the message $req : Loc \to Msg$ has one parameter of type *Loc*. The function *eq* compares two messages. For each message name *m* defined in *Msg*, a function $is\,Type_m : Mag \to Bool$ is defined which examines if a message term is constructed by the message name *m*.

The semantics of the data part (of a specification), denoted by $\mathbb{D}$, is defined the same way as in [13]. It should contain the *Bool* domain with distinct *T* and *F* constants, *Loc*, $\mathbb{C}$, and *Msg* domains.

### 3.2. Computed network terms

Let *D* denote a data sort; *u*, *v* and *d* range over closed and open data terms of sort *D*, respectively. Data terms are written as follows for the different sorts: *b* is of type *Bool*, $\mathfrak{m}$ is of type *Msg*, $\ell$ is of type *Loc*, and $\mathcal{C}$ is of type $\mathbb{C}$. Let $d[d_1/d_2]$ denote substitution of $d_2$ by $d_1$ in the data term *d*; this can be extended to computed network terms. Let $\mathcal{A}$ denote a countably infinite set of process names which are used as recursion variables in recursive specifications. This set can be split into two disjoint subsets $\mathcal{A}_p$ and $\mathcal{A}_n$. Without loss of generality we assume that process names and messages have exactly one parameter.

The transmission of network constraints into labels allows one to treat the so-called computed network terms, introduced in [3], as prefixed terms, so that the previous two-level syntax of *CNT* collapses to one:

$$
\begin{aligned}
t ::= \; & 0 \mid \beta.t \mid t + t \mid [b]t \diamond t \mid \textstyle\sum_{d:D} t \mid A(d)\,, A(d:D) \overset{def}{=} t \mid [\![t]\!]_\ell \mid \\
& t \mid t \mid t \lfloor\!\lfloor t \mid t \parallel t \mid (\nu \ell)t \mid \tau_m(t) \mid \partial_m(t)
\end{aligned}
$$

0 defines a deadlock process. The prefix operator in $\beta.t$ denote a process which performs $\beta$ and then behaves as *t*. The action $\beta$ can be of two types:

- $rcv(\mathfrak{m})$ and $snd(\mathfrak{m})$ actions, denoted by $\alpha$, which model the protocol receive and send actions respectively. They model the interaction of a protocol with its underlying *MAC* layer;
- $(\mathcal{C}, nrcv(\mathfrak{m}))$, $(\mathcal{C}, nsnd(\mathfrak{m}, \ell))$ and $(\mathcal{C}, \tau)$ actions, denoted by $(\mathcal{C}, \eta)$, where the first two actions are called the network receive and send actions respectively. They model the interaction of multiple *MAC* layers in a MANET. An action $(\mathcal{C}, \eta)$ represents the behavior $\eta$ for the set of topologies specified by $\mathcal{C}$.

The process $t_1 + t_2$ behaves non-deterministically as $t_1$ or $t_2$. The conditional construct $[b]t_1 \diamond t_2$ behaves as $t_1$ when $\mathbb{D} \models b = T$ and as $t_2$ when $\mathbb{D} \models b = F$. The summation $\sum_{d:D} t$, which binds the name *d* to *t*, defines a non-deterministic choice among $t[u/d]$ for all closed $u \in D$. A process name is declared by $A(d : D) \overset{def}{=} t$, where $A \in \mathcal{A}$, and *d* is a variable name that may appear free in *t*, meaning that it is not within the scope of a sum operator in *t*. Computed network terms are considered modulo $\alpha$-conversion of bound names. The function *fn*, which returns the set of free names, is defined over computed network terms as usual. A term is closed if the set of its free names is empty. The deployment of a process *t* at a network address $\ell \neq ?$ is specified as $[\![t]\!]_\ell$, which defines a single-node MANET. The parallel composition $t_1 \parallel t_2$ defines two MANETs that communicate by local broadcast; if there is a connectivity between nodes of $t_1$ and $t_2$ they may communicate, otherwise the send/receive actions of $t_1$ and $t_2$ are interleaved. *CNT* borrows from the process algebra *ACP* [15] the operators *left merge*

($\llfloor$) and *communication merge* ($|$) to axiomatize parallel composition. Hiding $(\nu\ell)t$ conceals the activities of a node with the address $\ell$ by renaming this address to ? in network send/receive actions. For each message type $m : D \to Msg$, the operators $\tau_m(-)$ and $\partial_m(-)$ are defined; *Abstraction* $\tau_m(t)$ renames network send/receive actions over messages of type $m$ to $\tau$, and *Encapsulation* $\partial_m(t)$ forbids receiving messages of type $m$ and renames them to 0. We use $\tau_{\{m_1,\ldots,m_n\}}(t)$ and $\partial_{\{m_1,\ldots,m_n\}}(t)$ to denote $\tau_{m_1}(\ldots(\tau_{m_n}(t))\ldots)$ and $\partial_{m_1}(\ldots(\partial_{m_n}(t))\ldots)$ respectively. We will use MANET, network and computed network terms interchangeably.

A computed network term $t$ should be grammatically well-defined:

- If $t \equiv [\![t']\!]_\ell$, then $t'$ has no network prefix action $(\mathcal{C}, \eta)$, deployment $[\![]\!]$, parallel $\|$, left merge $\llfloor$, communication merge $|$, hiding $(\nu\ell)$, abstraction $\tau_m$, encapsulation $\partial_m$, and process name $A(d)$ such that $A \in \mathcal{A}_n$.
- If $t \equiv rcv(m(d)).t'$, then it should be in the context of a summation like $\sum_{d:D}$, where $m : D \to Msg$.
- If $t \equiv \alpha.t'$, then it should be in the context of a deployment operator.
- If $t \equiv A(d)$ where $A \in \mathcal{A}_p$, then it should be in the context of a deployment operator. Furthermore it should be defined by an equation like $A(d : D) \stackrel{def}{=} t'$ such that $t'$ has no network prefix action $(\mathcal{C}, \eta)$, deployment $[\![]\!]$, parallel $\|$, left merge $\llfloor$, communication merge $|$, hiding $(\nu\ell)$, abstraction $\tau_m$, encapsulation $\partial_m$, and process name $A'(d)$ such that $A' \in \mathcal{A}_n$. Moreover, each occurrence of $A$ should be in the context of an $\alpha$ prefix action in $t'$.
- If $t \equiv B(d)$ where $B \in \mathcal{A}_n$, then it should not be in the context of a deployment operator. Furthermore it should be defined by an equation like $B(d : D) \stackrel{def}{=} t'$ such that $t'$ is well-defined.

Intuitively a computed network is grammatically well-defined if processes deployed at a network address, called protocols, are defined by protocol action prefix, choice, summation, conditional, deadlock operators and process names. From now on we will only consider computer network terms that are well-defined. For example, $[\![X(A)]\!]_A \parallel [\![Y(B)]\!]_B$ where $X(adr : Loc) \stackrel{def}{=} snd(req(A)).X(adr)$ and $Y(adr : Loc) \stackrel{def}{=} \sum_{lx:Loc} rcv(req(lx)).snd(rep(adr, lx))$. $Y(adr)$ is a well-defined computed network term. The process name $X$ defines a protocol which sends *req* messages iteratively, while $Y$ receives a *req* and then sends a *rep* message.

### 3.3. Rooted branching computed network bisimilarity

Computed network terms are considered modulo rooted branching computed network bisimilarity [3]. To define this equivalence relation, we introduce the following notations:

- $\Rightarrow$ denotes the reflexive and transitive closure of unobservable actions:
  - $t \Rightarrow t$;
  - if $t \xrightarrow{(\mathcal{C},\tau)} t'$ for some arbitrary network constraint $\mathcal{C}$ and $t' \Rightarrow t''$, then $t \Rightarrow t''$.
- $t \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t'$ iff $t \xrightarrow{(\mathcal{C},\eta)} t'$ or $t \xrightarrow{(\mathcal{C}[\ell/?],\eta[\ell/?])} t'$ and $\eta$ is of the form $nsnd(\mathfrak{m}, ?)$ for some $\mathfrak{m}$.

Intuitively $t \Rightarrow t'$ expresses that after a number of topology changes, $t$ can behave like $t'$. Furthermore, an action like $(\{? \rightsquigarrow B\}, nsnd(req(?), ?))$ can be matched to an action like $(\{A \rightsquigarrow B\}, nsnd(req(A), A))$, which is its $\langle - \rangle$ counterpart.

**Definition 2.** A binary relation $\mathcal{R}$ on computed network terms is a branching computed network simulation if $t_1\mathcal{R}t_2$ and $t_1 \xrightarrow{(\mathcal{C},\eta)} t_1'$ implies that either:

- $\eta$ is of the form $nrcv(\mathfrak{m})$ or $\tau$, and $t_1'\mathcal{R}t_2$; or
- there are $t_2'$ and $t_2''$ such that $t_2 \Rightarrow t_2'' \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t_2'$, where $t_1\mathcal{R}t_2''$ and $t_1'\mathcal{R}t_2'$.

$\mathcal{R}$ is a branching computed network bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are branching computed network simulations. Two terms $t_1$ and $t_2$ are branching computed network bisimilar, denoted by $t_1 \simeq_b t_2$, if $t_1\mathcal{R}t_2$ for some branching computed network bisimulation relation $\mathcal{R}$.

**Definition 3.** Two terms $t_1$ and $t$ are *rooted branching computed network bisimilar*, written $t_1 \simeq_{rb} t_2$, if:

- $t_1 \xrightarrow{(\mathcal{C},\eta)} t_1'$ implies there is a $t_2'$ such that $t_2 \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t_2'$ and $t_1' \simeq_b t_2'$;
- $t_2 \xrightarrow{(\mathcal{C},\eta)} t_2'$ implies there is an $t_1'$ such that $t_1 \xrightarrow{\langle(\mathcal{C},\eta)\rangle} t_1'$ and $t_1' \simeq_b t_2'$.

Rooted branching computed network bisimilarity is an equivalence relation and constitutes a congruence with respect to the *CNT* operators; see [3]. Intuitively two computed network terms are equivalent if they send and receive a same set of messages for a set of topologies. However a receiving action which would not change the sending behavior of a node can be removed. Therefore, an only receiving MANET (after its first action) is equivalent to deadlock. It should be noted that a node like $[\![Y(B)]\!]_B$ is not branching bisimilar to the sending node $[\![Y'(B)]\!]_B$ where $Y'(adr : Loc) \stackrel{def}{=} \sum_{lx:Loc} snd(rep(adr, lx)).Y'(adr)$, since the latter sends iff it receives a request message while the former always sends.

**Table 1**
Axioms for choice, conditional and summation operators.

| | | | |
|---|---|---|---|
| $Ch_1$ | $0 + t = t$ | $Sum_1$ | $\sum_{d:D} t = t, d \notin fn(t)$ |
| $Ch_2$ | $t_1 + t_2 = t_2 + t_1$ | $Sum_2$ | $\sum_{d:D} t = \sum_{e:D} t[e/d]$ |
| $Ch_3$ | $t_1 + (t_2 + t_3) = (t_1 + t_2) + t_3$ | $Sum_3$ | $\sum_{d:D} t = \sum_{d:D} t + t[u/d]$ |
| $Ch_4$ | $t + t = t$ | $Sum_4$ | $\sum_{d:D}(t_1 + t_2) = \sum_{d:D} t_1 + \sum_{d:D} t_2$ |
| $Con_1$ | $[b]t_1 \diamond t_2 = t_1,\ \mathbb{D} \models b = T$ | $Con_2$ | $[b]t_1 \diamond t_2 = t_2,\ \mathbb{D} \models b = F$ |
| $Ch_5$ | $(\mathcal{C}, nsnd(\mathfrak{m}, ?)).t + \langle(\mathcal{C}, nsnd(\mathfrak{m}, ?))\rangle.t = \langle(\mathcal{C}, nsnd(\mathfrak{m}, ?))\rangle.t$ | | |
| $Ch_6$ | $(\mathcal{C}_1, \eta).t + (\mathcal{C}_2, \eta).t = (\mathcal{C}_1, \eta).t,\ \mathcal{C}_1 \subseteq \mathcal{C}_2$ | | |

### 3.4. Axioms

We define the behavior of operators through their axioms over closed terms, which are sound with respect to rooted branching computed network bisimilarity. The axioms of choice, conditional and summation operator are given in Table 1. The axioms $Ch_{1-4}$, $Con_{1-2}$ and $Sum_{1-4}$ are straightforward (cf. [16]). The axiom $Ch_5$ is new in our framework, denoting that a network send action originated from a node of which the address is unknown can be removed if there is the same action originating from a node with a known address. The axiom $Ch_6$ explains that if an action $\eta$ is possible for a set of topologies, then it is also possible for all subsets of this set.

Axioms for process names are given in Table 2. *Unfold* and *Fold* express existence and uniqueness of a solution for the equation $A(d : D) \stackrel{def}{=} t$, which correspond to the *Recursive Definition Principle* (*RDP*) and *Recursive Specification Principle* (*RSP*) in *ACP*. An occurrence of a process name $A$ in $t$ is called *guarded* if this occurrence is in the scope of an action prefix operator (not $(\mathcal{C}, \tau)$ prefix) and not in the scope of an abstraction operator [3]. $A$ is *guarded* in $t$ if every occurrence of $A$ in $t$ is guarded.

Axioms for deployment, left and communication merge, and parallel operators are given in Table 3. The axioms $Dep_{3-5}$, $Br$, $LM_{1-4}$ and $S_{1-3,5}$ are straightforward. $Dep_1$ expresses that when a protocol sends a message (denoted by *snd*), the message is sent into the network (denoted by *nsnd*), irrespective of underlying topology (expressed by $\{\}$). $Dep_2$ expresses that when a protocol receives a message (denoted by *rcv*), it should receive it from the network (denoted by *nrcv*) while it is connected to some sender whose address is unknown (expressed by $\{? \rightsquigarrow \ell\}$). It should be noted that $Dep_5$ satisfies the second and third well-definedness rules given in Section 3.2.

The axioms $Sync_{1-3}$ explain the synchronization of two MANETs. The sending MANET $(\mathcal{C}_1, nsnd(\mathfrak{m}_1, \ell)).t_1$ can communicate with the receiving MANET $(\mathcal{C}_2, nrcv(\mathfrak{m}_2)).t_2$, if the receiving addresses (denoted by $\mathcal{C}_2$) are also connected to the sender $\ell$ (denoted by $\mathcal{C}_1 \cup \mathcal{C}_2[\ell/?]$). Likewise two receiving MANETs synchronize on a message when the receiving addresses of both MANETs are connected to the same unknown address (denoted by $\mathcal{C}_1 \cup \mathcal{C}_2$). Two sending MANETs cannot synchronize due to their signal collision. When a MANET is communicating through a $\tau$ action, it cannot be synchronized with another MANET, as indicated by axiom $S_4$.

We return to the example at the end of Section 3.2. The behavior of $[\![X(A)]\!]_A \parallel [\![Y(B)]\!]_B$ can be calculated as follows:

$$[\![X(A)]\!]_A \parallel [\![Y(B)]\!]_B = [\![X(A)]\!]_A \mathbin{\lfloor\!\lfloor} [\![Y(B)]\!]_B + [\![Y(B)]\!]_B \mathbin{\lfloor\!\lfloor} [\![X(A)]\!]_A + [\![X(A)]\!]_A \mid [\![Y(B)]\!]_B$$

$$[\![X(A)]\!]_A = (\{\}, nsnd(req(A), A)).[\![X(A)]\!]_A$$

$$[\![Y(B)]\!]_B = \sum_{lx:Loc} (\{? \rightsquigarrow B\}, nrcv(req(lx))).[\![snd(rep(B, lx)).Y(B)]\!]_B$$

$$[\![X(A)]\!]_A \mathbin{\lfloor\!\lfloor} [\![Y(B)]\!]_B = (\{\}, nsnd(req(A), A)).[\![X(A)]\!]_A \parallel [\![Y(B)]\!]_B$$

$$[\![Y(B)]\!]_B \mathbin{\lfloor\!\lfloor} [\![X(A)]\!]_A = \sum_{lx:Loc} (\{? \rightsquigarrow B\}, nrcv(req(lx))).[\![snd(rep(B, lx)).Y(B)]\!]_B \parallel [\![X(A)]\!]_A$$

$$[\![X(A)]\!]_A \mid [\![Y(B)]\!]_B = (\{A \rightsquigarrow B\}, nsnd(req(A), A)).[\![X(A)]\!]_A \parallel [\![snd(rep(B, A)).Y(B)]\!]_B$$

The axioms of hiding and encapsulation are given in Table 4. The *hiding* operator $(\nu\ell)\_$ conceals the address of a node with the address $\ell$ from external observers. Therefore, the behavior of a hidden node deploying process $X$, $(\nu C)[\![X(C)]\!]_C$, is $(\{\}, nsnd(req(?), ?)).(\nu C)[\![X(C)]\!]_C$. Then the behavior of $[\![X(A)]\!]_A \parallel (\nu C)[\![X(C)]\!]_C$, by application of axioms $Dep_{1,2}$, $Res_2$, $Br$, $LM_1$, $Sync_1$ and $Ch_5$, equals $(\{\}, nsnd(req(A), A)).[\![X(A)]\!]_A \parallel (\nu C)[\![X(C)]\!]_C$. This indicates that $[\![X(A)]\!]_A \parallel (\nu C)[\![X(C)]\!]_C = [\![X(A)]\!]_A$, since both are a solution of $Z \stackrel{def}{=} (\{\}, nsnd(req(A), A)).Z$ by axiom *Fold*. Intuitively, the hidden node $C$ does not change the behavior of $[\![X(A)]\!]_A$ from the point view of an external observer, since it assumes that the action of $C$ belongs to $A$.

**Table 2**
Axioms for process names.

| | |
|---|---|
| *Unfold* | $A(u) = t[u/d],\ A(d:D) \stackrel{def}{=} t$ |
| *Fold* | $\forall d : D \cdot t_1(d) = t_2[t_1(d_1)/A(d_1)]\cdots[t_1(d_n)/A(d_n)] \Rightarrow$ $t_1(d) = A(d),\ A(d:D) \stackrel{def}{=} t_2$ if $A$ is guarded in $t$ |

**Table 3**
Axioms for process names, deployment, left and communication merge, and parallel operators.

| | | | |
|---|---|---|---|
| $Dep_1$ | $[\![snd(\mathfrak{m}).t]\!]_\ell = (\{\}, nsnd(\mathfrak{m},\ell)).[\![t]\!]_\ell$ | $Dep_4$ | $[\![0]\!]_\ell = 0$ |
| $Dep_2$ | $[\![rcv(\mathfrak{m}).t]\!]_\ell = (\{? \rightsquigarrow \ell\}, nrcv(\mathfrak{m})).[\![t]\!]_\ell$ | $Dep_5$ | $\left[\!\!\left[\sum_{d:D} t\right]\!\!\right]_\ell = \sum_{d:D} [\![t]\!]_\ell$ |
| $Dep_3$ | $[\![t_1 + t_2]\!]_\ell = [\![t_1]\!]_\ell + [\![t_2]\!]_\ell$ | | |
| $Br$ | $t_1 \parallel t_2 = t_1 \mathbin{⫽} t_2 + t_2 \mathbin{⫽} t_1 + t_1 \mid t_2$ | $S_1$ | $t_1 \mid t_2 = t_2 \mid t_1$ |
| $LM_1$ | $(\mathcal{C},\eta).t_1 \mathbin{⫽} t_2 = (\mathcal{C},\eta).(t_1 \parallel t_2)$ | $S_2$ | $(t_1 + t_2) \mid t_3 = t_1 \mid t_3 + t_2 \mid t_3$ |
| $LM_2$ | $(t_1 + t_2) \mathbin{⫽} t_3 = t_1 \mathbin{⫽} t_3 + t_2 \mathbin{⫽} t_3$ | $S_3$ | $0 \mid t = 0$ |
| $LM_3$ | $0 \mathbin{⫽} t = 0$ | $S_4$ | $(\mathcal{C},\tau).t_1 \mid t_2 = 0$ |
| $LM_4$ | $\left(\sum_{d:D} t_1\right) \mathbin{⫽} t_2 = \sum_{d:D} t_1 \mathbin{⫽} t_2$ | $S_5$ | $\left(\sum_{d:D} t_1\right) \mid t_2 = \sum_{d:D} t_1 \mid t_2$ |
| $Sync_1$ | $(\mathcal{C}_1, nsnd(\mathfrak{m}_1,\ell)).t_1 \mid (\mathcal{C}_2, nrcv(\mathfrak{m}_2)).t_2 = [eq(\mathfrak{m}_1,\mathfrak{m}_2)](\mathcal{C}_1 \cup \mathcal{C}_2[\ell/?], nsnd(\mathfrak{m}_1,\ell)).t_1 \parallel t_2 \diamond 0$ | | |
| $Sync_2$ | $(\mathcal{C}_1, nrcv(\mathfrak{m}_1)).t_1 \mid (\mathcal{C}_2, nrcv(\mathfrak{m}_2)).t_2 = [eq(\mathfrak{m}_1,\mathfrak{m}_2)](\mathcal{C}_1 \cup \mathcal{C}_2, nrcv(\mathfrak{m}_1)).t_1 \parallel t_2 \diamond 0$ | | |
| $Sync_3$ | $(\mathcal{C}_1, nsnd(\mathfrak{m}_1,\ell_1)).t_1 \mid (\mathcal{C}_2, nsnd(\mathfrak{m}_2,\ell_2)).t_2 = 0$ | | |

**Table 4**
Axiomatization of hiding, abstraction and encapsulation operators.

| | | | |
|---|---|---|---|
| $Res_1$ | $(\nu\ell)(t_1 + t_2) = (\nu\ell)t_1 + (\nu\ell)t_2$ | $Res_3$ | $(\nu\ell)0 = 0$ |
| $Res_2$ | $(\nu\ell)(\mathcal{C},\eta).t = (\mathcal{C}[?/\ell], \eta[?/\ell]).(\nu\ell)t$ | $Res_4$ | $(\nu\ell)\sum_{d:D} t = \sum_{d:D}(\nu\ell)t$ |
| $Ecp_1$ | $\partial_m((\mathcal{C}, nsnd(\mathfrak{m},\ell)).t) = (\mathcal{C}, nsnd(\mathfrak{m},\ell)).\partial_m(t)$ | | |
| $Ecp_2$ | $\partial_m((\mathcal{C}, nrcv(\mathfrak{m})).t) = [\neg isType_m(\mathfrak{m})](\mathcal{C}, nrcv(\mathfrak{m})).\partial_m(t) \diamond 0$ | | |
| $Abs_1$ | $\tau_m((\mathcal{C},\eta).t) = (\mathcal{C}, \tau_m(\eta)).\tau_m(t)$ | | |
| $Abs_2$ | $\tau_m(t_1 + t_2) = \tau_m(t_1) + \tau_m(t_2)$ | $Ecp_3$ | $\partial_m(t_1 + t_2) = \partial_m(t_1) + \partial_m(t_2)$ |
| $Abs_3$ | $\tau_m(0) = 0$ | $Ecp_4$ | $\partial_m(0) = 0$ |
| $Abs_4$ | $\tau_m\left(\sum_{d:D} t\right) = \sum_{d:D} \tau_m(t)$ | $Ecp_5$ | $\partial_m\left(\sum_{d:D} t\right) = \sum_{d:D} \partial_m(t)$ |
| $T_1$ | $(\mathcal{C},\eta).((\mathcal{C}', nrcv(\mathfrak{m})).t + t) = (\mathcal{C},\eta).t$ | | |
| $T_2$ | $(\mathcal{C},\eta).((\mathcal{C}',\tau).(t_1 + t_2) + t_2) = (\mathcal{C},\eta).(t_1 + t_2)$ | | |

The axiom $Abs_1$ renames $\eta$ actions carrying messages of type $m$ to $\tau_m(\eta)$, which is defined as follows:

$$\tau_m(nrcv(\mathfrak{m})) = if\,(isType_m(\mathfrak{m}), \tau, nrcv(\mathfrak{m}))$$
$$\tau_m(nsnd(\mathfrak{m},\ell)) = if\,(isType_m(\mathfrak{m}), \tau, nsnd(\mathfrak{m},\ell))$$

The axiom $Ecp_2$ explains that the *encapsulation* operator renames network receive actions of messages of type $m$ to 0. For example,

$$\begin{aligned}\partial_{req}([\![X(A)]\!]_A \parallel [\![Y(B)]\!]_B) &= (\{\}, nsnd(req(A), A)).\partial_{req}([\![X(A)]\!]_A \parallel [\![Y(B)]\!]_B) \\ &\quad + (\{A \rightsquigarrow B\}, nsnd(req(A), A)).\partial_{req}([\![X(A)]\!]_A \parallel [\![snd(rep(B, A)).Y(B)]\!]_B)\end{aligned}$$

Axiom $T_1$ removes a receive action that does not affect the behavior of a network, while $T_2$ removes a $\tau$ action which preserves the behavior of a network after some topology changes. The remaining axioms in this table are straightforward.

## 4. Case study: a simple routing protocol

We consider a simple routing protocol, which is similar to AODV [17] in its basic concepts. In a MANET, each node can communicate with other nodes indirectly by exploiting a routing protocol. In these protocols, all nodes act as router and relay messages to the next hop for an intended destination. To this aim, each node keeps the address of the next hop for some destinations in a *routing table*. When a node needs to transmit data to a destination, it first retrieves in its routing table the address of the next hop to which it should send data. If the next hop is unknown, it initiates the *route discovery*

$$
\begin{array}{l}
Init(adr, dst : Loc) \stackrel{def}{=} \\
\quad \sum_{lx:Loc} [\neg eq(lx, ?) \wedge \neg eq(lx, adr) \wedge \neg eq(lx, dst)] snd(data(lx)).0 \diamond 0 \\
Mid(nx : Loc, adr : Loc) \stackrel{def}{=} \\
\quad [\neg(eq(nx, ?))]( \\
\quad\quad \sum_{lx:Loc} rcv(data(lx)). \\
\quad\quad\quad [eq(lx, adr)] snd(data(nx)).Mid(nx, adr) \diamond Mid(nx, adr) + \\
\quad\quad \sum_{lx:Loc} rcv(error(lx)). \\
\quad\quad\quad [eq(lx, nx)] snd(error(adr)).RtDy(adr, ?) \diamond Mid(nx, adr) + \\
\quad\quad snd(error(adr)).RtDy(adr, ?) + \\
\quad\quad \sum_{lx:Loc} rcv(req(lx)).snd(rep(adr, lx)).Mid(nx, adr)) + \\
\quad \diamond \ (RtDy(adr, ?) + \\
\quad\quad \sum_{lx:Loc} rcv(req(lx)).RtDy(adr, lx)) \\
RtDy(adr : Loc, src : Loc : Bool) \stackrel{def}{=} \\
\quad snd(req(adr)). \\
\quad\quad (\sum_{lx:Loc} \sum_{ly:Loc} rcv(rep(lx, ly)).( \\
\quad\quad\quad [eq(ly, adr)] \\
\quad\quad\quad\quad [\neg eq(src, ?)] snd(rep(adr, src)).Mid(lx, adr) \diamond Mid(lx, adr) \\
\quad\quad\quad \diamond RtDy(adr, src)) + RtDy(adr, src)) \\
Dst(adr : Loc) \stackrel{def}{=} \\
\quad \sum_{lx:Loc} rcv(req(lx)).snd(rep(adr, lx), 0)).Dst(adr) + \\
\quad \sum_{lx:Loc} rcv(data(lx)).[eq(lx, adr)]0 \diamond Dst(adr).
\end{array}
$$

**Fig. 5.** The specifications of the initiator, middle and destination processes.

*process*. In this process, the node broadcasts a $req(dst, adr)$ message, where *adr* is the address of the node itself, to ask its neighbors whether they know a route to a node with address *dst*. On receiving a $req(dst, src)$ message, a node examines its routing table; if it knows a route to the destination, it replies by sending a $rep(dst, adr, src)$ message, where *adr* and *src* are the addresses of the receiving and requesting nodes respectively. Otherwise it rebroadcasts *req* by substituting *src* for its own address. Each node, upon receiving the $rep(dst, nx, adr)$ message, updates its routing table by setting the address of the next hop for *dst* to *nx*, and relays the message to its requesting neighbor, if it is not the initiator of route discovery. When a node with address *adr* detects that its route to *dst* is not valid any more due to a link break-down, it broadcasts the message $error(dst, adr)$ to inform its neighbors that it cannot be used as a router to *dst*. If a node that uses the address *nx* as the next hop for transmitting data to *dst* receives $error(dst, nx)$, then it erases this routing record in its routing table, and informs its neighbors by replacing *nx* by its own address.

### 4.1. Protocol specification

To ease the process of verification, we decompose the specification of the protocol into three processes, namely initiator, middle and destination. The initiator node delivers its data to a middle node, to route its data to the destination. The middle nodes find a route to the destination node, update this route with regard to topology changes, and carry data along a route. The destination node replies to requests, and receives data destined for it.

We specify a network composed of four nodes, where one nodes deploys the initiator process, two nodes the middle process, and one node the destination process. Since we focus on finding a route to a specific *dst*, we model the routing table with a variable *nx* of sort *Loc*, and remove *dst* from the parameter list of messages like *req*, *rep* and *error*. The specifications of the initiator, middle and destination processes, called *Init*, *Mid* and *Dst* respectively, are given in Fig. 5. Process *RtDy* specifies the route discovery process; *src* denotes the node for which the route discovery process was initiated and should be replied to.

### 4.2. Protocol analysis

The most fundamental error in routing protocol operations is failure to route correctly. The correct operation of MANET routing protocols can be defined as follows [18]: *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 path from the source node to the destination node*. A situation which violates the above property is a routing loop, meaning that somewhere along the path from the source to its destination a packet can enter a forwarding circle. We are going to examine whether our simple routing protocol is loop-free. To this aim, we encode the processes in mCRL2, to derive the *CLTS* of the MANET:

$$\mathcal{M}_0 \equiv \partial_{\{req,rep,error,data\}}([\![Init(A, D)]\!]_A \parallel [\![Mid(?, B)]\!]_B \parallel [\![Mid(?, C)]\!]_C \parallel [\![Dst(D)]\!]_D).$$

With regard to the fourth well-definedness rule, and by application of axioms $Dep_{1-5}$, $Con_{1,2}$ and *Fold*, for every *CNT* term $[\![t(d)]\!]_\ell$, there is a network name $A(d:D) \overset{def}{=} t'$, where $A \in \mathcal{A}_n$, such that $[\![t(d)]\!]_\ell = A(d)$. To encode $\mathcal{M}_0$, we first derive equivalent network names for $[\![Init(\ell,\ell')]\!]_\ell$, $[\![Mid(\ell',\ell,b)]\!]_\ell$, and $[\![Dst(\ell)]\!]_\ell$, namely $Init_n(\ell,\ell')$, $Mid_n(\ell',\ell,b)$, and $Dst_n(\ell)$.

The only difference between parallel composition of mCRL2 and *CNT* is on their synchronization part; in mCRL2, two actions are synchronized if they agree on the number and values of their parameters, while in *CNT* two actions are synchronized if they agree on the message part, while some calculations are performed on their network constraints (see axioms $Sync_{1-3}$). To model the local broadcast communication of *CNT* by the parallel composition of mCRL2, we define a set of actions $nsnd_i, nrcv_j : \mathbb{C} \times Msg \times Loc$, where $0 \le i, \le n$, $1 \le j \le n$ with $n$ the number of nodes. The action $nrcv_i(\{\ell \rightsquigarrow \ell_1,\ldots,\ell_i\},\mathtt{m},\ell)$ denotes that the message $\mathtt{m}$, when sent by the node with address $\ell$, can be received by $i$ nodes with addresses $\ell_1,\ldots,\ell_i$, because they are connected to the sender. And $nsnd_i(\{\ell \rightsquigarrow \ell_1,\ldots,\ell_i\},\mathtt{m},\ell)$ denotes that the node with address $\ell$ sends the message $\mathtt{m}$ while $i$ nodes with addresses $\ell_1,\ldots,\ell_i$ are connected to it and consequently can receive $\mathtt{m}$. To model the network constraint calculations, each $(\{\},nsnd(\mathtt{m},\ell)).t$ and $(\{?\rightsquigarrow\ell\},nrcv(\mathtt{m})).t$ (resulting from the axioms $Dep_{1,2}$ in the previous step) is encoded as

$$
\begin{aligned}
&(\{\},nsnd(\mathtt{m},\ell)).t:\\
&\quad nsnd_0(\{\},\mathtt{m},\ell).t\,+\\
&\quad \sum_{\ell_1:Loc}([\neg eq(\ell,\ell_1)]nsnd_1(\{\ell\rightsquigarrow\ell_1\},\mathtt{m},\ell).t \diamond 0+\\
&\quad \sum_{\ell_2:Loc}([\neg eq(\ell,\ell_2)\wedge\neg eq(\ell_1,\ell_2)]nsnd_2(\{\ell\rightsquigarrow\ell_1,\ell_2\},\mathtt{m},\ell).t \diamond 0\,+\\
&\quad \cdots +\\
&\quad \sum_{\ell_n:Loc}([\neg eq(\ell,\ell_n)\wedge\cdots\wedge\neg eq(\ell_{n-1},\ell_n)]nsnd_n(\{\ell\rightsquigarrow\ell_1,\ldots,\ell_n\},\mathtt{m},\ell).t \diamond 0)\ldots))\\
&(\{?\rightsquigarrow\ell\},nrcv(\mathtt{m})).t:\\
&\quad \sum_{\ell_1:Loc}([\neg eq(\ell,\ell_1)]nrcv_1(\{\ell_1\rightsquigarrow\ell\},\mathtt{m},\ell_1).t \diamond 0+\\
&\quad \sum_{\ell_2:Loc}([\neg eq(\ell,\ell_2)\wedge\neg eq(\ell_1,\ell_2)]nrcv_2(\{\ell_1\rightsquigarrow\ell,\ell_2\},\mathtt{m},\ell_1).t \diamond 0\,+\\
&\quad \cdots +\\
&\quad \sum_{\ell_n:Loc}([\neg eq(\ell,\ell_n)\wedge\cdots\wedge\neg eq(\ell_{n-1},\ell_n)]nrcv_n(\{\ell_1\rightsquigarrow\ell,\ell_2,\ldots,\ell_n\},\mathtt{m},\ell_1).t \diamond 0)\ldots))
\end{aligned}
$$

where the sum, choice, conditional and action prefix operators are mCRL2 constructs with the same semantics as in *CNT*. A *CNT* term $t$ with its network receive and send actions encoded as above is denoted by $\Im(t)$. The *CNT* term $\partial_{\widetilde{M}}(t_1 \parallel \ldots \parallel t_n)$, where $\widetilde{M}$ is the set of all messages, is modeled by the mCRL2 operators renaming $\rho$, allow $\nabla$, communication $\Gamma$, and parallel $\|$, where $\rho_{\{a\to b\}}$ renames the action name $a$ to $b$, $\nabla_{\{a\}}$ renames all actions except $a$ to deadlock, and $\Gamma_{\{a|b\to c\}}$ renames synchronized actions $a$ and $b$ to $c$:

$$
\rho_{\{nsnd_0\to nsnd\}}(\nabla_{\{nsnd_0,nsnd\}}(\Gamma_{\{nsnd_1|nrcv_1\to nsnd,\ldots,nsnd_n|\underbrace{nrcv_n|\ldots|nrcv_n}_{n\text{ items}}\to nsnd\}}(\Im(t_1)\parallel\ldots\parallel\Im(t_n)))).
$$

Thus, the encoding of $\mathcal{M}_0$ is achieved by setting $n$ to 4, and $t_1$, $t_2$, $t_3$ and $t_4$ to $Init_n(A,D)$, $Mid_n(?,B)$, $Mid_n(?,C)$, and $Dst_n(D)$ in the above formula. The labeled transition system resulting from this encoding contains labels of the form $nsnd(\mathcal{C},\mathtt{m},\ell)$. Since only middle nodes look for a route to the destination, the loop can only occur between the middle nodes $B$ and $C$. Therefore we can examine the existence of a loop by the following regular $\mu$-calculus formula [19]:

$$
\begin{aligned}
&\langle true^\star\rangle\langle nsnd(\{B\rightsquigarrow C\},data(C),B)\rangle\\
&\qquad\langle nsnd(\{C\rightsquigarrow B\},data(B),C)\rangle true
\end{aligned}
$$

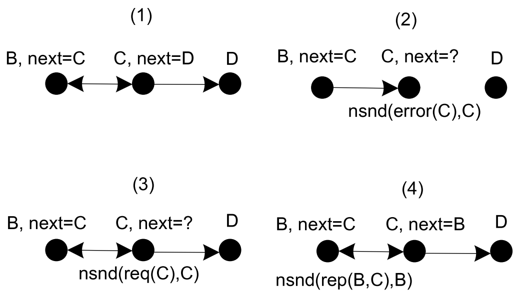


**Fig. 6.** A scenario leading to a loop formation in the simple routing protocol.

where $\langle true^{\star}\rangle$ at the start of the formula denotes any system trace, and *true* at the end of the formula any state. The CADP model checker confirms that the above property holds, and returns the following execution:

$$\mathcal{M}_0 \xrightarrow{nsnd(\{B\rightsquigarrow C\},req(B),B)} \mathcal{M}_1 \xrightarrow{nsnd(\{C\rightsquigarrow D\},req(C),C)} \mathcal{M}_2 \xrightarrow{nsnd(\{D\rightsquigarrow C\},rep(D,C),D)} \mathcal{M}_3$$
$$\mathcal{M}_3 \xrightarrow{nsnd(\{C\rightsquigarrow B\},rep(C,B),C)} \mathcal{M}_4 \xrightarrow{nsnd(\{\},error(C),C)} \mathcal{M}_5 \xrightarrow{nsnd(\{C\rightsquigarrow B,D\},req(C),C)} \mathcal{M}_6$$
$$\mathcal{M}_6 \xrightarrow{nsnd(\{B\rightsquigarrow C\},rep(B,C),B)} \mathcal{M}_7 \xrightarrow{nsnd(\{A\rightsquigarrow B\},data(B),A)} \mathcal{M}_8$$
$$\mathcal{M}_8 \xrightarrow{nsnd(\{B\rightsquigarrow C\},data(C),B)} \mathcal{M}_9 \xrightarrow{nsnd(\{C\rightsquigarrow B\},data(B),C)} \mathcal{M}_8 \ldots$$

From this one can derive the following scenario during which a loop is formed. Let $B$ have a route to $D$ through $C$ (Fig. 6(1)), and then the link between $C$ and $D$ goes down. Next $B$ loses the *error* message because of a temporary link failure between $C$ and $B$ (Fig. 6(2)). Then the link between $C$ and $B$ becomes valid and $C$ requests a path to $D$ (Fig. 6(3)). Finally $B$ replies and a loop is formed (Fig. 6(4)). This scenario complies with the scenario explained in [20]. However, there the model is verified against a specific mobility scenario, while in our approach the model is verified against many instances of mobility scenarios at the same time. Therefore, as explained in Section 2, we can derive mobility scenarios leading to the (undesired) property.

A solution to prevent loop-formation is assigning a sequence number to each route, to track changes in the underlying topology (and using hop counts to choose the shorter path). When there is a topology change, the sequence number is incremented. Thus the protocol is refined as follows: each node sends its *req* by appending its known sequence number (for the destination), to indicate the freshness of the route required. Each node also keeps the sequence number for each destination in its routing table, and replies to a request only if its sequence number is at least as much as the one in the request message. When a route expires, the node should keep the incremented sequence number for that destination, to remember the sequence number for which it should initiate the request, as remarked in [20]. We have also experienced this in model checking, as otherwise a loop is formed.

We revised our code by assigning a sequence number (and a hop count) to each route (as shown in Section 5.4). To keep the state space finite, we specified that a *Mid* process can only detect a link breakage once, since it causes an increase in the sequence number. By model checking we are sure that the protocol is correct for scenarios leading to one link breakage with three middle nodes. We will verify the correctness of the improved protocol for an arbitrary number of link breakages and number of middle nodes in Section 5.4, using a symbolic verification technique.

## 5. Symbolic verification

To prove the correctness of a communication protocol, it is common to prove a network composed of a number of nodes each deploying the protocol — referred to as the *implementation* — equivalent to a more abstract description — the *specification* — of the desired external behavior.

We rephrase the question whether the implementation of a MANET and its specification are equivalent in terms of proof obligations on relations between data objects. This technique is based on the cones and foci method [6]. A restricted class of *CNT* specifications, called linear computed network equations, are considered, in which the states are data objects. To prove equivalence of an implementation and a specification, given in this linear format, a *state mapping* between the data objects of the implementation and specification is given. The proof is completed by showing that the state mapping constitutes a branching computed network bisimulation.

### *5.1. Linear computed network equations and invariants*

A linear computed network equation (LCNE) is a computed network term consisting of only action prefix, summation and conditional operators; it does not contain any parallel, encapsulation, abstraction or hiding operators. An LCNE is basically a vector of data parameters together with a list of condition, action and effect triples, describing for each state under which condition an action may happen and what is its effect on the vector of data parameters. Each computed network term can be transformed into an LCNE using the axioms (cf. [21]). In this paper we do not discuss the algorithm transforming a network specification into an LCNE, but will only consider one example in Section 5.3.

Without loss of generality, we assume that each message constructor has exactly one parameter. Let the set of (concrete) actions be $Act^c = \{nsnd(m(-), \ell), nrcv(m(-)) | \forall m : D_m \to Msg, \forall \ell \in Loc\}$, ranged over by $\eta(-)$.

**Definition 4.** A linear computed network equation is a *CNT* specification of the form

$$A(d : D) \stackrel{def}{=} \sum_{\eta : Act^c \cup \{\tau\}} \sum_{e:E} [h_\eta(d, e)](\mathcal{C}_\eta(d, e), \eta(f_\eta(d, e))).A(g_\eta(d, e)) \diamond 0$$

where $h_\eta : D \times E \to Bool$, $\mathcal{C}_\eta : D \times E \to \mathbb{C}$, $f_\eta : D \times E \to D_m$ and $g_\eta : D \times E \to D$ for each $\eta \in Act^c \cup \{\tau\}$.

The LCNE in Definition 4 has exactly one *CLTS* as its solution (modulo strong bisimilarity). In this *CLTS*, the states are data elements $d : D$, where $D$ may be a Cartesian product of $n$ data types, i.e. $(d_1, \ldots, d_n)$, the transition labels are the network send and receive actions of messages parameterized with data, and the transition constraints are network constraints parameterized with data. The LCNE expresses that state $d$ can send/receive message $\eta(f_\eta(d, e))$ for the set of topologies specified by $\mathcal{C}_\eta(d, e)$ to end up in state $g_\eta(d, e)$ under the condition that $h_\eta(d, e)$ is true.

**Definition 5.** A mapping $\mathcal{I} : D \to Bool$ is an invariant for an LCNE, written as in Definition 4, if for all $\eta \in Act^c \cup \{\tau\}$, $d : D$ and $e : E$,

$$\mathcal{I}(d) \wedge h_\eta(d, e) \Rightarrow \mathcal{I}(g_\eta(d, e)).$$

Invariants can be used to characterize the set of reachable states of an LCNE. Namely, if $\mathcal{I}(d)$ and it is possible to perform $\eta(f_\eta(d, e))$ (since $h_\eta(d, e)$ holds), then $\mathcal{I}$ holds in the resulting state $g_\eta(d, e)$.

### *5.2. Equivalence checking by using state mappings*

The system implementation and specification, both given in linear format, are branching computed network bisimilar, if there exists a state mapping $\phi$ between them which satisfies the transfer conditions of Definition 2. An invariant $\mathcal{I}$ can be imposed; then the transfer conditions only need to hold in states where $\mathcal{I}$ is true, and consequently equivalence between implementation and specification is only guaranteed to hold in states where $\mathcal{I}$ is true.

To allow infinite sequences of $\tau$-transitions in the implementation, we leave the abstraction operator $\tau_{\widetilde{M}}$ around it, to ensure that it has a unique solution. The set of communications over $\widetilde{M}$ is defined by $I_{\widetilde{M}}$ as

$$\{nsnd(\mathcal{C}, \mathfrak{m}, \ell), nrcv(\mathcal{C}, \mathfrak{m}) | \exists m \in \widetilde{M} \cdot isType_m(\mathfrak{m})\}.$$

Let $\langle\eta\rangle$ denote $\eta$ or, $\eta[\ell/?]$ and $\eta$ is of the form $nsnd(\mathfrak{m}, ?)$. Depending on the value of $\langle\eta\rangle$, for any binary relation $\odot$, $r_\eta(e, d) \odot r'_{\langle\eta\rangle}(e, d')$ iff $r_\eta(e, d) \odot r'_\eta(e, d')$ or $r_\eta(e, d)[\ell/?] \odot r'_{\eta[\ell/?]}(e, d')$.

**Proposition 1.** *Let the LCNE Imp be of the form*

$$Imp(d : D) \stackrel{def}{=} \sum_{\eta \in Act^c \cup \{\tau\}} \sum_{e:E} [h_\eta(d, e)](\mathcal{C}_\eta(d, e), \eta(f_\eta(d, e))).Imp(g_\eta(d, e)) \diamond 0$$

*Furthermore, let the LCNE Spec be of the form*

$$Spec(d' : D') \stackrel{def}{=} \sum_{\eta \in Act^c \setminus I_{\widetilde{M}}} \sum_{e:E} [h'_\eta(d', e)](\mathcal{C}'_\eta(d', e), \eta(f'_\eta(d', e))).Spec(g'_\eta(d', e)) \diamond 0$$

*Let $\mathcal{I} : D \to Bool$ be an invariant for Imp, and $\phi : D \to D'$ a state mapping. If for all $\eta \in Act^c \setminus I_{\widetilde{M}}$ and $\eta_\tau \in I_{\widetilde{M}}$, $\phi$ satisfies the following conditions:*

1. *$\forall e : E(h_{\eta_\tau}(d, e) \Rightarrow \phi(d) = \phi(g_{\eta_\tau}(d, e)))$;*
2. *$\forall e : E, h_\eta(d, e)$ implies that either $\eta$ is a receive action such that $\phi(d) = \phi(g_\eta(d, e))$, or $h'_{\langle\eta\rangle}(\phi(d), e)$ holds for some $\langle\eta\rangle$ such that $f_\eta(d, e) = f'_{\langle\eta\rangle}(\phi(d), e)$, $\mathcal{C}'_{\langle\eta\rangle}(\phi(d), e) \subseteq \mathcal{C}_\eta(d, e)$, and $\phi(g_\eta(d, e)) = g'_{\langle\eta\rangle}(\phi(d), e)$;*

3. *$\forall e : E$, $h'_\eta(\phi(d), e)$ implies that either $\eta$ is a receive action such that $\phi(d) = g'_\eta(\phi(d), e)$, or there exists $d^*$ such that $d \xrightarrow{\eta_{\tau_1}}_{\mathcal{C}_1} \dots \xrightarrow{\eta_{\tau_n}}_{\mathcal{C}_n} d^*$, where $\eta_{\tau_1}, \dots, \eta_{\tau_n} \in I_{\widetilde{M}}$, and for some $\langle\eta\rangle$, $h_{\langle\eta\rangle}(d^*, e)$ holds with $f_{\langle\eta\rangle}(d^*, e) = f'_\eta(\phi(d), e)$, $\mathcal{C}_{\langle\eta\rangle}(d^*, e) \subseteq \mathcal{C}'_\eta(\phi(d), e)$, and $\phi(g_{\langle\eta\rangle}(d^*, e)) = g'_\eta(\phi(d), e))$;*

*then for all $d : D$ with $\mathcal{I}(d)$, $\tau_{\widetilde{M}}(Imp(d)) \simeq_b Spec(\phi(d))$.*

See Appendix for the proof. Since each state of the specification defines the external behavior of the implementation with regard to any possible topology changes, the mapped state of the implementation should not be changed by $\tau$-transitions (which may be triggered due to some topology changes), as implied by the first criterion. And each state of the implementation has the same observable behavior as its mapped state in the specification, directly or after some topology changes, as implied by the second and third criterion.

Due to mobility of nodes, MANET protocols usually contain mechanisms to examine if a node connection to some other node exists or not. For instance, a node may examine whether it is still connected to its next hop for a destination in a routing protocol, or to its leader in a leader election protocol. Such mechanisms are modeled by non-deterministic behavior in the protocol specification, which restarts some part of the process (like route discovery in a routing protocol). Due to such mechanisms, in each state of the implementation, the observable behavior may change after a set of $\tau$-transitions. On the other hand, since we assume arbitrary mobility for MANET nodes, each state of the specification defines the behavior of a MANET for any possible topology change. Therefore, we lack a collection of so-called focus points [6,7]: states in the implementation that can be matched to some state in the specification with the same observable behavior.

For example, to show that $\forall n : Nat \cdot N(n) \simeq_b M(n)$, where

$$N(n : Nat) \stackrel{def}{=} [n \geq 1](\{\}, nsnd(data(B), A)).N(n+1) \diamond 0 + [n \geq 1](\{\}, nsnd(data(B), ?)).N(n+2) \diamond 0$$

$$M(b : Bool) \stackrel{def}{=} [eq(b, T)](\{\}, nsnd(data(B), A)).M(b) \diamond 0$$

it suffices to show that $\phi(n) = if(n \geq 1, T, F)$ satisfies the second and third conditions of Proposition 1 (as there is no abstraction):

- When $n \geq 1$ holds, two actions $\eta_1 \equiv nsnd(data(-), A)$ and $\eta_2 \equiv nsnd(data(-), ?)$ are possible. For the first action, $f_{\eta_1}(n) = B$, $\mathcal{C}_{\eta_1}(n) = \{\}$, and $g_{\eta_1}(n) = n + 1$. Since $\phi(n) = T$, $h'_{\eta_1}(T)$ while $f_{\eta_1}(n) = f'_{\eta_1}(T)$, $\mathcal{C}'_{\eta_1}(T) \subseteq \mathcal{C}_{\eta_1}(n)$, and $\phi(g_{\eta_1}(n)) = g'_{\eta_1}(T)$. For the second action, $f_{\eta_2}(n) = B$, $\mathcal{C}_{\eta_2}(n) = \{\}$, and $g_{\eta_2}(n) = n + 2$. The only action of $M$ is again matched to this action, since $\langle nsnd(data(-), ?)\rangle = nsnd(data(-), A)$, while $f_{\eta_2}(n) = f'_{\langle\eta_2\rangle}(T)$, $\mathcal{C}'_{\langle\eta_2\rangle}(T) \subseteq \mathcal{C}_{\eta_2}(n)$, and $\phi(g_{\eta_2}(n)) = g'_{\langle\eta_2\rangle}(T)$.
- The only action of $M$ when $eq(\phi(n \geq 1), T))$ is $\eta \equiv nsnd(data(-), A)$, and the same action is enabled in $N$ when $n \geq 1$, with the same parameter and network constraint.

### *5.3. Linearization of uniform MANETs*

In practice a MANET often consists of an arbitrary set of similar nodes: each node is identified by a unique network address, and deploys the same protocols. In this section we show how our symbolic verification approach can be exploited to verify such networks. To this aim, we first provide a general recursive specification for MANETs with similar nodes, and then derive a linear computed network equation as a solution of the recursive specification, using the *CNT* axioms, data axioms and induction. The derived linear equation is strongly bisimilar to the original recursive equation.

Without loss of generality, we assume that each message constructor has exactly one parameter. We assume that each process $P(\ell, d)$ is defined using a linear process equation (LPE) [22] of the form:

$$\begin{aligned} &P(\ell : Loc, d : D) \stackrel{def}{=} \\ &\quad \textstyle\sum_{m \in Msg} \sum_{e : E_m} [h_{m_s}(\ell, d, e)] snd(m(f_{m_s}(\ell, d, e))).P(\ell, g_{m_s}(\ell, d, e)) \diamond 0 + \\ &\quad\quad [h_{m_r}(\ell, d, e)] rcv(m(f_{m_r}(\ell, d, e))).P(\ell, g_{m_r}(\ell, d, e)) \diamond 0 \end{aligned} \tag{1}$$

where $h_{m_s/m_r} : Loc \times D \times E_m \to Bool$, $f_{m_s/m_r} : Loc \times D \times E_m \to D_m$ and $g_{m_s/m_r} : Loc \times D \times E_m \to D$ for each $m \in Msg$.

As we do not want to fix the addresses of nodes in the MANET beforehand, we use two auxiliary data sorts: *LocList* which is a list of network addresses of nodes, and similar to the approach of [9], *DTable* which is a table indexed by network addresses, where each entry maintains the state of the node at the corresponding network address. We also exploit for each $m \in Msg$ an auxiliary data sort $EList_m$, which is a list of elements of sort $E_m$, the auxiliary data type used in functions of messages (see Eq. (1)).

The sort *LocList* is defined below. Lists are generated from the empty list *empL* and *add*, which places a new address in the list. The function *has* examines if an element belongs to the list; *include* examines if the first list is included in the second list; *remove* removes an address from the list; *head* returns the first element of the list; *size* returns the length of the list; *nodup* examines if the list has no duplicated item; and *eq* compares two lists.

To increase readability, we write binary functions in infix manner, and use symbols $\emptyset$, $\rhd$, $\in$, $\subseteq$, $\setminus$, $|\,|$ and $\ell l[0]$ for *empL*, *add*, *has*, *include*, *remove*, *size* and $head(\ell l)$, respectively. The data sort $EList_m$ for $m \in Msg$ is defined in the same way as *LocList*, but using the constant $empE_m$.

$$
\begin{array}{lll}
sort & LocList & \\
func & empL : \to LocList & \\
 & add : Loc \times LocList \to LocList & \\
map & has : Loc \times LocList \to Bool & \\
 & include, eq : LocList \times LocList \to Bool & \\
 & remove : LocList \times Loc \to LocList & \\
 & head : LocList \to Loc & \\
 & size : LocList \to Nat & \\
 & nodup : LocList \to Bool & \\
var & \ell l, \ell l_1, \ell l_2 : LocList, \ell, \ell_1, \ell_2 : Loc & \\
rew & has(\ell, empL) = F & LA_1 \\
 & has(\ell_1, add(\ell_2, \ell l)) = if(eq(\ell_1, \ell_2), T, has(\ell_1, \ell l)) & LA_2 \\
 & include(empL, \ell l) = T & LA_3 \\
 & include(add(\ell, \ell l_1), \ell l_2) = has(\ell, \ell l_2) \wedge include(\ell l_1, \ell l_2) & LA_4 \\
 & remove(empL, \ell) = empL & LA_5 \\
 & remove(add(\ell_1, \ell l), \ell_2) = if(eq(\ell_1, \ell_2), remove(\ell l, \ell_2), add(\ell_1, remove(\ell l, \ell_2))) & LA_6 \\
 & head(add(\ell, \ell l)) = \ell & LA_7 \\
 & size(empL) = 0 & LA_9 \\
 & size(add(\ell, \ell l)) = size(\ell l) + 1 & LA_{10} \\
 & nodup(empL) = T & LA_{11} \\
 & nodup(add(\ell, \ell l)) = \neg has(\ell, \ell l) \wedge nodup(\ell l) & LA_{12} \\
 & eq(\ell l_1, \ell l_2) = include(\ell l_1, \ell l_2) \wedge include(\ell l_2, \ell l_1) & LA_{13}
\end{array}
$$

Tables are generated from the constant *empT* and an operation *upd*, which places a new entry in the table. The function *get* gets an entry from the table using its index. The function $upd\_all_{g_m}(\ell \rhd \ell l, e \rhd el, dt)$ updates the list of entries $\ell \rhd \ell l$ in the table using the function $g_m : Loc \times D \times E_m \to D$; the entry $\ell$ is updated by $g_m(\ell, get(\ell, dt), e)$, which uses the network address $\ell$, the previous value at the entry, and an auxiliary value $e$. Intuitively this function is helpful to update a set of receiver nodes that communicate with a sender over message $m$. Similarly the function $and\_all_{h_m, f_m, f'_m}(\ell_1 \rhd \ell l, \ell_2, e_2, e_1 \rhd el, dt)$ examines a Boolean expression on a list of entries $\ell_1 \rhd \ell l$ using functions $h_m : Loc \times D \times E_m \to Bool$ and $f_m, f'_m : Loc \times D \times E_m \to D_m$; for each entry $\ell_1$, it examines if $h_m(\ell_1, get(\ell_1, dt), e_1)$ evaluates to true and if $f'_m(\ell_1, get(\ell_1, dt), e_1)$ is equal to $f_m(\ell_2, get(\ell_2, dt), e_2)$. Intuitively this function is helpful to examine if a set of nodes can synchronize with each other upon receiving a message of type $m$, i.e., whether the conditions of their actions are true (examined by $h_m$) and their message parameters are equal to each other (examined by $f_m, f'_m$).

$$
\begin{array}{lll}
sort & DTable & \\
func & empT : \to DTable & \\
 & upd : Loc \times D \times DTable \to DTable & \\
map & get : Loc \times DTable \to D & \\
 & upd\_all_{g_m} : LocList \times EList_m \times DTable \to DTable & \\
 & and\_all_{h_m, f_m, f'_m} : LocList \times Loc \times E_m \times EList_m \times DTable \to Bool & \\
var & \ell, \ell_1, \ell_2 : Loc, \ell l : LocList, & \\
 & d : D, dt : DTable, & \\
 & e, e_1, e_2 : E_m, el : EList_m & \\
rew & get(\ell_1, upd(\ell_2, d, dt)) = if(eq(\ell_1, \ell_2), d, get(\ell_1, dt)) & TA_1 \\
 & upd\_all_{g_m}(empL, el, dt) = dt & TA_2 \\
 & upd\_all_{g_m}(add(\ell, \ell l), add(e, el), dt) = & \\
 & \quad upd(\ell, g_m(\ell, get(\ell, dt), e), upd\_all_{g_m}(\ell l, el, dt)) & TA_3 \\
 & and\_all_{h_m, f_m, f'_m}(empL, \ell, e, el, dt) = T & TA_4 \\
 & and\_all_{h_m, f_m, f'_m}(add(\ell_1, \ell l), \ell_2, e_2, add(e_1, el), dt) = & \\
 & \quad and(h_m(\ell_1, get(\ell_1, dt), e_1), and(eq(f_m(\ell_2, get(\ell_2, dt), e_2), & \\
 & \qquad f'_m(\ell_1, get(\ell_1, dt), e_1)), and\_all_{h_m, f_m, f'_m}(\ell l, \ell_2, e_2, el, dt))) & TA_5
\end{array}
$$

Axioms $TA_{2-5}$ are schematic and can be defined for all functions $g_{m_s/m_r}$, $h_{m_s/m_r}$, $f_{m_s/m_r}$ in Eq. (1) for any $m \in Msg$.

In the remainder we write $dt[\ell]$ instead of $get(\ell, dt)$. The following network recursive specification puts nodes deploying process $P$ at network addresses of $\ell l$ in parallel.

$$
Manet(\ell l : LocList, dt : DTable) \stackrel{def}{=} [eq(\ell l, \emptyset)]0 \diamond [\![P(\ell l[0], dt[\ell l[0]])]\!]_{\ell l[0]} \parallel Manet(\ell l \setminus \ell l[0], dt). \tag{2}
$$

$$
\begin{array}{l}
Mid(nx : Loc, hp : Nat, sq : Nat, adr : Loc) \stackrel{def}{=} \\
\quad [\neg(eq(nx, ?))]( \\
\quad\quad \sum_{lx:Loc} rcv(data(lx)).[eq(lx, adr)] \\
\quad\quad\quad snd(data(nx)).Mid(nx, hp, sq, adr) \\
\quad\quad\quad\quad \diamond Mid(nx, hp, sq, adr) + \\
\quad\quad \sum_{lx:Loc} rcv(error(lx)). \\
\quad\quad\quad [eq(lx, nx)]snd(error(adr)).RtDy(sq + 1, adr, ?) \\
\quad\quad\quad\quad\quad\quad\quad \diamond Mid(nx, hp, sq, adr) + \\
\quad\quad snd(error(adr)).RtDy(sq + 1, adr, ?) + \\
\quad\quad \sum_{lx:Loc} \sum_{sx:Nat} rcv(req(lx, sx)). \\
\quad\quad\quad\quad [sq \geq sx]snd(rep(adr, lx, sq, hp)).Mid(nx, hp, sq, adr) \\
\quad\quad\quad\quad\quad \diamond RtDy(sx, adr, lx)) \\
\quad \diamond \;\; (RtDy(sq, adr, ?) + \\
\quad\quad \sum_{lx:Loc} \sum_{sx:Nat} rcv(req(lx, sx)).RtDy(\max(sx, sq), adr, lx)) \\
\\
RtDy(sq : Nat, adr : Loc, src : Loc) \stackrel{def}{=} \\
\quad snd(req(adr, sq)). \\
\quad\quad (\sum_{lx:Loc} \sum_{ly:Loc} \sum_{sx:Nat} \sum_{hpx:Nat} rcv(rep(lx, ly, sqx, hpx)).( \\
\quad\quad\quad [eq(ly, adr) \wedge sx \geq sq] \\
\quad\quad\quad\quad ([\neg eq(src, ?)]snd(rep(adr, src, sqx, hpx + 1)).Mid(lx, hpx + 1, sqx, adr) \\
\quad\quad\quad\quad \diamond Mid(lx, hpx + 1, sqx, adr)) \\
\quad\quad\quad \diamond RtDy(sq, adr, src)) \\
\quad\quad + RtDy(sq, adr, src)
\end{array}
$$

**Fig. 7.** The revised specifications of the middle process.

Below we present the core lemma of this section. It gives an expansion of *Manet*, where all operators for parallelism have been removed. The resulting network has the list $\ell l$ and the table $dt$ as parameters. In essence, the complexity of the computed network *Manet* is now encoded using the list and table operations.

Lemma 1 says that in the network $X$, the node with network address $k \in \ell l$ may send the message $m$, parameterized by data from this node, if it is ready to send (as indicated by $h_{m_s}(k, dt[k], e)$) to a list $\ell_s$ (without duplicates) of receiver nodes with addresses in $\ell l \setminus k$ that are all ready to receive such a message (examined by $and\_all_{h_{m_r}, f_{m_s}, f_{m_r}}$). Table entries with indices in $\ell_s$ and $k$ are updated as a result of this communication (using $upd\_all_{g_{m_r}}$). The function $\mathcal{C}(\ell, \ell_s) = \{\ell \rightsquigarrow \ell' | \ell' \in \ell_s\}$ specifies the network constraint for this behavior of the network, indicating there is a communication link from $\ell$ to each node in $\ell_s$. Nodes in the network $X$ may also receive a message $m$ from an unknown address ?; the receiving nodes must have network addresses in $\ell_s$, where $\ell_s \subseteq \ell l \wedge \neg eq(\ell_s, \emptyset) \wedge nodup(\ell_s)$, and must be ready to receive such a message (examined by $and\_all_{h_{m_r}, f_{m_r}, f_{m_r}}$). All table entries with indices in $\ell_s$ are updated as a result of this receive action (using $upd\_all_{g_{m_r}}$).

**Lemma 1.** *The MANET Manet as defined in Eqs.* (1) *and* (2) *is a solution for the MANET X in Eq.* (3) *below.*

$$
\begin{array}{l}
X(\ell l : LocList, dt : DTable) \stackrel{def}{=} \\
\quad \sum_{m \in Msg} \sum_{k:Loc} \sum_{\ell_s:LocList} \sum_{e:E_m} \sum_{el:EList_m} \\
\quad\quad [k \in \ell l \wedge \ell_s \subseteq \ell l \setminus k \wedge nodup(\ell_s) \wedge |\ell_s| = |el| \wedge \\
\quad\quad\; h_{m_s}(k, dt[k], e) \wedge and\_all_{h_{m_r}, f_{m_s}, f_{m_r}}(\ell_s, k, e, el, dt)] \\
\quad\quad\quad (\mathcal{C}(k, \ell_s), nsnd(m(f_{m_s}(k, dt[k], e)), k)). \\
\quad\quad\quad\quad X(\ell l, upd(k, g_{m_s}(k, dt[k], e), upd\_all_{g_{m_r}}(\ell_s, el, dt))) \diamond 0 + \\
\quad \sum_{m \in Msg} \sum_{\ell_s:LocList} \sum_{el:EList_m} \\
\quad\quad [\ell_s \subseteq \ell l \wedge \neg eq(\ell_s, \emptyset) \wedge nodup(\ell_s) \wedge |\ell_s| = |el| \wedge \\
\quad\quad\; and\_all_{h_{m_r}, f_{m_r}, f_{m_r}}(\ell_s, \ell_s[0], el[0], el, dt)] \\
\quad\quad\quad (\mathcal{C}(?, \ell_s), nrcv(m(f_{m_r}(\ell_s[0], dt[\ell_s[0]], el[0])))). \\
\quad\quad\quad\quad X(\ell l, upd\_all_{g_{m_r}}(\ell_s, el, dt)) \diamond 0.
\end{array}
\tag{3}
$$

See [14] for the proof. The following Composition Theorem is a corollary of Lemma 1 and the axiom *Fold*.

**Theorem 1.** $Manet(\ell l, dt) = X(\ell l, dt)$.

### *5.4. Verification of the improved routing protocol*

The revised versions of the processes of Fig. 5, which exploit sequence numbers to trace the freshness of routes and hop counts to choose the shortest paths, are specified in Figs. 7 and 8.

Then the linear formats of these specifications are given in Figs. 9 and 10 (see [14] for explanations how the linear formats are derived). In each summand of the LPEs (and LCNEs later), we only present the parameters whose values are changed: $d/x$ denotes that the parameter $x$ is assigned the data term $d$. Moreover, $b$ and $\neg b$ denote $eq(b, T)$ and $eq(b, F)$ respectively. In these specifications, all request and reply messages carry the sequence number of the path they request for or reply to, while reply messages also carry the hop count of the path. When a process broadcasts the message *error* to inform its neighbors that it cannot be used as a router, it increments its sequence number, which will be used later in the route discovery process.

$$
\begin{array}{l}
Dst(sq:Nat, adr:Loc) \stackrel{def}{=} \\
\quad \sum_{lx:Loc} \sum_{sx:Nat} rcv(req(lx, sx)). \\
\qquad snd(rep(adr, lx, \max(sqx, sq), 0)).Dst(\max(sqx, sq), adr) \\
\quad \sum_{lx:Loc} rcv(data(lx)).[eq(lx, adr)]0 \diamond Dst(sq, adr).
\end{array}
$$

**Fig. 8.** The revised specification of destination process.

$$
\begin{array}{l}
Init(s:Nat, adr, dst:Loc) \stackrel{def}{=} \\
\quad \sum_{lx:Loc} [eq(s,0) \wedge \neg eq(lx,?) \wedge \neg eq(lx, adr) \wedge \neg eq(lx, dst)] \\
\qquad snd(data(lx)).Init(1/s) \diamond 0 \\
Dst(s:Nat, src:Loc, sq:Nat, adr:Loc) \stackrel{def}{=} \\
\quad \sum_{lx:Loc} \sum_{sx:Nat} [eq(s,0)] rcv(req(lx, sx)). \\
\qquad Dst(1/s, \max(sx, sq)/sq, lx/src) \diamond 0 + \\
\quad [eq(s,1)] snd(rep(adr, src, sq, 0)).Dst(0/s) \diamond 0 + \\
\quad \sum_{lx:Loc} [eq(s,0)] rcv(data(lx)).Dst(if(eq(lx, adr), 2, s)/s) \diamond 0.
\end{array}
$$

**Fig. 9.** The linearized equations of the initiator and destination processes.

$$
\begin{array}{l}
Mid(s:Nat, nx, src:Loc, sq, hp:Nat, adr:Loc, dih:Bool) \stackrel{def}{=} \\
\quad [dih] snd(data(nx)).Mid(F/dih) \diamond 0 + \\
\quad \sum_{lx:Loc} [eq(s,0) \wedge \neg eq(nx,?) \wedge \neg dih] \\
\qquad rcv(data(lx)).Mid(if(eq(lx, adr), T, dih)/dih) \diamond 0 + \\
\quad [(eq(s,0) \wedge eq(nx,?)) \vee s \geq 3] \\
\qquad snd(req(adr, sq)).Mid(4/s, if(eq(s,0), ?, src)/src) \diamond 0 + \\
\quad \sum_{lx:Loc} \sum_{sx:Nat} [\neg dih \wedge eq(s,0)] rcv(req(lx, sx)). \\
\qquad Mid(if(sx > sq \vee eq(nx,?), 3, 2)/s, lx/src, \max(sq, sx)/sq) \diamond 0 + \\
\quad [eq(s,2)] snd(rep(adr, src, sq, hp)).Mid(0/s) \diamond 0 + \\
\quad \sum_{lx:Loc} \sum_{dx:Loc} \sum_{sx:Nat} \sum_{hx:Nat} [eq(s,4)] \\
\qquad rcv(rep(lx, dx, sx, hx)). \\
\qquad\quad Mid(if(eq(dx, adr) \wedge sx \geq sq, if(\neg eq(src,?), 2, 0), s)/s, \\
\qquad\quad\; if(eq(dx, adr) \wedge sx \geq sq, lx, nx)/nx, \\
\qquad\quad\;\; if(eq(dx, adr) \wedge sx \geq sq, sx, sq)/sq, \\
\qquad\quad\;\;\; if(eq(dx, adr) \wedge sx \geq sq, (hx+1), hp)/hp) \diamond 0 + \\
\quad [eq(s,1) \vee (\neg dih \wedge eq(s,0) \wedge \neg eq(nx,?))] snd(error(adr)). \\
\qquad Mid(3/s, ?/src, (sq+1)/sq) \diamond 0 + \\
\quad \sum_{lx:Loc} [\neg dih \wedge eq(s,0) \wedge \neg eq(nx,?)] \\
\qquad rcv(error(lx)).Mid(if(eq(lx, nx), 1, s)/s) \diamond 0
\end{array}
$$

**Fig. 10.** The linearized equation of the middle process.

$$
\begin{array}{l}
Routing(n, N:Nat, fin:Bool) \stackrel{def}{=} \\
\quad [(\neg fin \wedge n > 1) \vee eq(n,0)] \\
\qquad (\{\}, nsnd(data(?), ?)).Routing(T/fin) \diamond 0 + \\
\quad \sum_{h:Nat} [(\neg fin \wedge n > 1 \wedge h < N) \vee (eq(n,0) \wedge h \leq N)] \\
\qquad (\{\}, nsnd(data(?), ?)).Routing(h/n) \diamond 0 + \\
\quad [\neg fin \wedge eq(n,1)](\{\}, nsnd(data(B), ?)).Routing(T/fin) \diamond 0
\end{array}
$$

**Fig. 11.** The desired external behavior.

Therefore, a node that has not received an error message on a route for a destination, cannot reply to a request message, since its sequence number is less than the sequence number of the request message. The *dih* parameter in process *Mid* is introduced during the linearization process to indicate when data is held by the node.

The desired external behavior of a MANET running the routing protocol is given by the process *Routing* in Fig. 11. The intuition behind this specification is: when data is held by a middle node ($n \geq 1$) and there is no routing loop on its route to the destination, the distance that the data message should pass to reach the destination, specified by $n$, is at most $N$, where $N$ is the number of middle nodes. However, when there is a movement or an error among the nodes including the next node on the route (and the next hop is not the destination, i.e. $n > 1$), the route and consequently the distance to pass may change. This change is specified by arbitrary changes of $n$ to a value less than $N$ (since the middle node holding the data would not participate in the route discovery of its next hop, the number of middle nodes participating in the route discovery is at most $N - 1$). For a network with only the known address $B$, the data message begins its journey from some initiator (that itself does not know any route to the destination) with $n = 0$, until it reaches the destination $B$, in the meantime moving between (middle nodes with) unknown addresses. If a next hop loses the data or the destination receives the data, there is no further data message. The Boolean variable *fin* is false as long as the initiator or a middle node holds the data. In any state, either the data is safely transferred to the next hop (while the distance of the next hop may change in case the next hop is not the destination), or the next hop may lose it (and consequently *fin* is updated to true). Due to arbitrary mobility of nodes, a route is always found from any middle node to the destination. Therefore, the desired external behavior specifies that data always reaches its destination, unless it is lost on the way.

We are going to prove that the parallel composition of a node with the initiator process, a finite number of nodes deploying the middle process, specified by *nMid* using Eq. (2), and a node with the destination process, behaves like *Routing*:

$$nMid(\ell l : LocList, \xi t : \Xi Table) \stackrel{def}{=} [eq(\ell l, \emptyset)]0 \diamond (\llbracket Mid(\xi t[\ell l[0]]) \rrbracket_{\ell l[0]} \parallel nMid(\ell l \setminus \ell l[0], \xi t))$$

where $nodup(\ell l)$, $\Xi : Nat \times Loc^2 \times Nat^2 \times Loc \times Bool$, and $\Xi Table$ is a table containing elements of sort $\Xi$. Let $\xi \in \Xi$ represent the sequence $\langle s, nx, src, hp, sq, adr, dih \rangle$. $get_{dih}(\ell, \xi t)$ returns the $dih$ element of the entry with index $\ell$ in the table $\xi t : \Xi Table$, and $upd_{dih}(\ell, b, \xi t)$ updates such an element. We use $dih_i$ or $\xi t[i].dih$ to denote $get_{dih}(i, \xi t)$. Our goal is to derive the following equation ( Theorem 2):

$$(\{\}, \tau).Routing(0, |\ell l|, F) = (\{\}, \tau).\tau_{\widetilde{M}_2}(\partial_{\widetilde{M}_1}((\nu A) \llbracket Init(0, A, B) \rrbracket_A \parallel (\nu \ell l) nMid(\ell l, \xi t) \parallel \llbracket Dst(0, ?, 0, B) \rrbracket_B)) \tag{4}$$

where $\widetilde{M}_1 = \{req, rep, error, data\}$, $\widetilde{M}_2 = \{req, rep, error\}$, $(\nu \ell l)$ abbreviates $(\nu \ell_1) \ldots (\nu \ell_n)$ for all $\ell_1, \ldots, \ell_n \in \ell l, A, B \notin \ell l$, and for all $i \leq |\ell l|$, the $i^{th}$ entry of table $\xi t$ is $\langle 0, ?, ?, 0, 0, i, F \rangle$. The initial $\tau$ actions specify the initial route discoveries of middle nodes. To prove Eq. (4) regarding Lemma 2, we exploit the symbolic verification technique to show that:

$$Routing(0, |\ell l|, fin) \simeq_b \tau_{\widetilde{M}_2}(InitnMidDst(0, \ell l, \xi t, 0, ?, 0))$$

where

$$InitnMidDst(s_A : Nat, \ell l : LocList, \xi t : \Xi Table, s_B : Nat, src_B : Loc, sq_B : Loc) \stackrel{def}{=}$$

$$\sum_{lx:Loc} \sum_{ls:LocList} \left[ eq(s_A, 0) \wedge ls \subseteq \ell l \wedge \bigwedge_{i \in ls} (eq(s_i, 0) \wedge \neg eq(nx_i, ?) \wedge \neg dih_i) \right] \tag{1}$$
$$(\{\}, nsnd(data(?), ?)).InitnMidDst(1/s_A, \forall_{i \in ls} if(eq(lx, i), T, dih_i)/dih_i) \diamond 0 +$$

$$\sum_{k:Loc} \sum_{ls:LocList} \left[ k \in \ell l \wedge ls \subseteq (B \triangleright \ell l \setminus k) \wedge dih_k \bigwedge_{i \in ls \setminus B} (eq(s_i, 0) \wedge \neg eq(nx_i, ?) \wedge \neg dih_i \wedge (B \in ls \Rightarrow eq(s_B, 0))) \right] \tag{2}$$
$$(if(B \in ls, \{? \rightsquigarrow B\}, \{\}), nsnd(data(if(eq(nx_k, B), B, ?)), ?)). InitnMidDst(F/dih_k, \forall_{i \in ls \setminus B} if(eq(nx_k, i), T, dih_i)/dih_i, if(B \in ls \wedge eq(nx_k, B), 2, s_B)/s_B) \diamond 0 +$$

$$\sum_{k:Loc} \sum_{ls:LocList} \left[ k \in \ell l \wedge ls \subseteq (B \triangleright \ell l \setminus k) \wedge ((eq(s_k, 0) \wedge eq(nx_k, ?)) \vee s_k \geq 3) \bigwedge_{i \in ls \setminus B} (\neg dih_i \wedge eq(s_i, 0) \wedge (B \in ls \Rightarrow eq(s_B, 0))) \right] \tag{3}$$
$$(if(B \in ls, \{? \rightsquigarrow B\}, \{\}), nsnd(req(?, sq_k), ?)). InitnMidDst(4/s_k, if(eq(s_k, 0), ?, src_k)/src_k, \forall_{i \in ls \setminus B}( if(sq_k > sq_i \vee eq(nx_i, ?), 3, 2)/s_i, k/src_i, \max(sq_i, sq_k)/sq_i), if(B \in ls, 1, s_B)/s_B, if(B \in ls, max(sq_k, sq_B), sq_B)/sq_B, if(B \in ls, k, src_B/src_B)) \diamond 0+$$

$$\sum_{k:Loc} \sum_{ls:LocList} \left[ k \in \ell l \wedge ls \subseteq (\ell l \setminus k) \wedge eq(s_k, 2) \bigwedge_{i \in ls \setminus B} eq(s_i, 4) \right] \tag{4}$$
$$(\{\}, nsnd(rep(?, ?, sq_k, hp_k), ?)).InitnMidDst(0/s_k, ( \forall_{i \in ls} if(eq(src_k, i) \wedge sq_k \geq sq_i, if(\neg eq(src_i, ?), 2, 0), s_i)/s_i, if(eq(src_k, i) \wedge sq_k \geq sq_i, k, nx_i)/nx_i, if(eq(src_k, i) \wedge sq_k \geq sq_i, sq_k, sq_i)/sq_i, if(eq(src_k, i) \wedge sq_k \geq sq_i, hp_k + 1, hp_i)/hp_i)) \diamond 0+$$

$$\sum_{ls:LocList}\left[eq(s_B,1)\wedge ls\subseteq \ell l\wedge \neg eq(ls,\emptyset)\bigwedge_{i\in ls} eq(s_i,4)\right] \tag{5}$$
$$(\{\}, nsnd(rep(B,?,sq_B,0),B)).InitnMidDst((\forall_{i\in ls} if(eq(src_B,i)\wedge sq_B\geq sq_i, if(\neg eq(src_i,?),2,0),s_i)/s_i,$$
$$if(eq(src_B,i)\wedge sq_B\geq sq_i, B, nx_i)/nx_i,$$
$$if(eq(src_B,i)\wedge sq_B\geq sq_i, sq_B, sq_i)/sq_i,$$
$$if(eq(src_B,i)\wedge sq_B\geq sq_i, 0, hp_i)/hp_i), 0/s_B)\diamond 0+$$

$$\sum_{k:Loc}\sum_{ls:LocList}\left[k\in \ell l\wedge ls\subseteq(\ell l\setminus k)\wedge(eq(s_k,1)\right. \tag{6}$$
$$\left.\vee(\neg dih_k\wedge eq(s_k,0)\wedge\neg eq(nx_k,?)))\bigwedge_{i\in ls}(\neg dih_i\wedge eq(s_i,0)\wedge\neg eq(nx_i,?))\right]$$
$$(\{\}, nsnd(error(?),?)).InitnMidDst(3/s_k, ?/src_k, (sq_k+1)/sq_k,$$
$$\forall_{i\in ls} if(eq(k,nx_i),1,s_i)/s_i)\diamond 0$$

where $\bigwedge_{i\in\ell_s}$ examines a Boolean expression on, and $\forall_{i\in\ell_s}$ updates, a set of entries, implemented like the functions $and\_all_{h_m,f_m,f'_m}$ and $upd\_all_{g_m}$, respectively. For instance, $\bigwedge_{i\in ls} eq(s_i,4)$ and $\forall_{i\in ls} if(eq(k,nx_i),1,s_i)/s_i)$ are equal to $and\_all(ls,\xi t)$ and $upd\_all(ls,k,\xi t)$ respectively, where:

$$and\_all(\emptyset,\xi t)=T$$
$$and\_all(\ell\triangleright\ell l,\xi t)=eq(\xi t[\ell].s,4)\wedge and\_all(\ell l,\xi t)$$
$$upd\_all(\emptyset,\ell',\xi t)=empT$$
$$upd\_all(\ell\triangleright\ell l,\ell',\xi t)=upd_s(\ell, if(eq(\ell',\xi t[\ell].nx),1,\xi t[\ell].s), upd\_all(\ell l,\ell',\xi t)).$$

**Lemma 2.**

$$InitnMidDst(s_A,\ell l,\xi t,s_B,src_B,sq_B)=\partial_{\widetilde{M}_1}((\nu A)[\![Init(s_A,A,B)]\!]_A \parallel (\nu\ell l)nMid(\ell l,\xi t)\parallel [\![Dst(s_B,src_B,sq_B,B)]\!]_B).$$

**Proof.** We first expand $nMid(\ell l,\xi t)$ by application of Composition Theorem 1 (see [14]), and then $\partial_{\widetilde{M}_1}((\nu A)[\![Init(s_A,A,B)]\!]_A \parallel (\nu\ell l)nMid(\ell l,\xi t)\parallel [\![Dst(s_B,src_B,sq_B,B)]\!]_B)$ by application of parallel, hiding, and encapsulation and $LA_{1-13}$ axioms. We conclude the proof by application of *Fold*. □

We introduce the state mapping $\phi : Nat\times LocList\times \Xi Table\times Nat\times Loc\times Nat\rightarrow Nat^2\times Bool$, where $\phi(s_A,\ell l,\xi t,s_B,src_B,sq_B)=(n,N,fin)$ is defined:

$$n=if(\neg fin, if(eq(s_A,0),0,\exists_{i\in\ell l}\cdot dih_i\Rightarrow hp_i),\text{ any value})$$
$$N=|\ell l|$$
$$fin=\bigwedge_{i\in\ell l}\neg dih_i\wedge eq(s_A,1)$$

As long as data is held by some node in the network ($\neg fin$), the value of 0 for $n$ denotes that the data is held by the initiator (that does not know any route to the destination) while the value $n\geq 1$ denotes that the data is held by a middle node and so this value specifies the distance that data message should pass to reach the destination. Therefore, when data is not held by the initiator ($\neg eq(s_A,0)$), its value is the hop count of middle node holding the data. The maximum distance that the data message can pass equals the number of middle nodes. Since *fin* implies no further data transmission, it becomes true if the middle nodes and the initiator do not have the data. The values of $s_B$, $src_B$ and $sq_B$ do not affect $\phi$, so we write $\phi(s_A,\ell l,\xi t)$ instead of $\phi(s_A,\ell l,\xi t,s_B,src_B,sq_B)$.

Invariants of $InitnMidDst(s_A,\ell l,\xi t,s_B,src_B,sq_B)$ are:

$$\mathcal{I}_1\equiv eq(s_i,0)\vee eq(s_i,1)\vee eq(s_i,2)\vee eq(s_i,3)\vee eq(s_i,4)\vee eq(s_i,5)$$
$$\mathcal{I}_2\equiv (eq(s_B,0)\vee eq(s_B,1)\vee eq(s_B,2))\wedge(eq(s_A,0)\vee eq(s_A,1))$$
$$\mathcal{I}_3\equiv eq(nx_i,j)\vee eq(nx_i,?)$$
$$\mathcal{I}_4\equiv dih_i\Rightarrow eq(s_A,1)\wedge\forall_{k\in\ell l\wedge\neg eq(k,i)}\neg dih_k$$
$$\mathcal{I}_5\equiv dih_i\Rightarrow eq(s_i,0)\wedge\neg eq(nx_i,?)$$
$$\mathcal{I}_6\equiv eq(nx_i,j)\wedge eq(s_j,0)\Rightarrow\neg eq(nx_j,?)$$
$$\mathcal{I}_7\equiv eq(nx_i,j)\wedge s_i\leq 2\Rightarrow sq_j\geq sq_i\wedge hp_i>0$$

$$\mathcal{I}_8 \equiv eq(nx_i, j) \wedge eq(sq_i, sq_j) \wedge s_i \leq 2 \Rightarrow eq(hp_i, hp_j + 1)$$
$$\mathcal{I}_9 \equiv eq(nx_i, B) \Leftrightarrow eq(hp_i, 1) \wedge sq_B \geq sq_i$$
$$\mathcal{I}_{10} \equiv eq(s_i, 3) \vee eq(s_i, 4) \Leftrightarrow eq(nx_i, ?)$$
$$\mathcal{I}_{11} \equiv s_i \leq 2 \Rightarrow \neg eq(src_i, ?)$$

where $i, j \in \ell l$ such that $\neg eq(i, j)$.

Invariants $\mathcal{I}_{1-3}$ define the ranges of variables. Intuitively $\mathcal{I}_4$ explains that only one middle node or the initiator can hold the data, and $\mathcal{I}_5$ explains this is when that node has a route to the destination ($\neg eq(nx, ?)$) and stays in the state 0. Each next hop always has a route unless it is involved in another route discovery (when $\neg eq(s_j, 0)$), as stated by $\mathcal{I}_6$. Invariants $\mathcal{I}_{7,8}$ imply that on a route from a middle node to the destination, either the sequence numbers increase, or the sequence numbers are equal (denoting to a stable route) and the hop counts decrease. When a middle node is directed connected to the destination, its hop count is 1, as explained by $\mathcal{I}_9$. The existence of a route in the node $i$ is inferred by the condition $s_i \leq 2$, as implied by $\mathcal{I}_{10}$. By $\mathcal{I}_{11}$, a node may send a reply to a node with address $src_i$ if it is involved in a route discovery.

**Lemma 3.** *$\mathcal{I}_{1-11}$ are invariants of InitnMidDst$(s_A, \ell l, \xi t, s_B, src_B, sq_B)$.*

**Proof.** We only prove invariants $\mathcal{I}_7$ and $\mathcal{I}_8$ together; the others can be proved with a similar argumentation. We start from a state with $eq(s_i, 0) \wedge eq(s_j, 0) \wedge eq(nx_i, j) \wedge eq(sq_i, sq_j) \wedge eq(hp_i + 1, hp_j) \wedge eq(nx_j, k) \wedge \neg eq(i, k)$. According to the values of $dih_i$ and $dih_j$, three cases can be considered. We examine the activities of node $i$ and $j$ in these states, to trace how $sq_i, sq_j$ and $hp_i, hp_j$ are changed. We use $x'$ to denote the updated value of $x$ in the next state:

- $dih_i$: In this state, according to $\mathcal{I}_4$, $\neg dih_j$. By summand (2), node $i$ may send a data message, and two cases can be distinguished. If $j \in ls$, a state with $\neg dih'_i \wedge dih'_j$ is reached, otherwise a state with $\neg dih'_i$ is reached. Both cases are examined later. By summand (6), node $j$ may send an error while $i \not\in ls$ (since $dih_i$), and a state with $eq(s'_j, 3) \wedge sq'_j = sq_j + 1$ is reached. From this state, only states with $eq(nx_i, j) \wedge sq_j > sq_i \wedge dih_i$ are reached, unless node $i$ sends data which is examined later. By summand (6), node $j$ may receive an error message from any node other than $i$ and a state with $eq(s'_j, 1)$. Again from this state, node $j$ may send an error message, and a state with $eq(s'_j, 3) \wedge sq'_j = sq_j + 1$ is reached as discussed before. By summand (3), node $j$ may receive a request from any node other than $i$. Depending on the value of the carried sequence number, a state with $eq(s'_j, 2)$ or $eq(s'_j, 3)$ is reached. In the former case, node $j$ can only send a reply message by summand (4), and then a state with $eq(s'_j, 0)$ is reached again. In the latter case, a state with $eq(s'_j, 3) \wedge sq'_j = sq_j + 1$ is reached as discussed before.
- $dih_j$: In this state, according to $\mathcal{I}_4$, it holds that $\neg dih_i$. By summand (2), node $j$ may send a data message, and only a state with $\neg dih_i \wedge \neg dih_j$ is reached (which is discussed later). By summand (6), node $i$ may send an error while $j \not\in ls$ (since $dih_j$), and a state with $eq(s'_i, 3) \wedge sq'_i = sq_i + 1$ is reached. From this state, only states with $eq(s_i, 0) \wedge \neg eq(nx_i, j)$ are reached, unless node $j$ sends data which is examined later. By summand (6), node $i$ may receive an error, but since it was from a node other than $j$, its state is not changed. By summand (3), node $i$ may receive a request from any node other than $j$. Depending on the value of the carried sequence number, a state with $eq(s'_i, 2)$ or $eq(s'_i, 3)$ is reached. In the former case, node $i$ can only send a reply message by summand (4), and then a state with $eq(s'_i, 0)$ is reached again. In the latter case, a state with $eq(s'_i, 3) \wedge sq'_i = sq_i + 1$ is reached, as discussed before.
- $\neg dih_i \wedge \neg dih_j$: By summands (3), (6), (2) and (1), the following cases need to be considered:
  - By summand (3), node $i$ may receive a request from any node other than $j$ (since $\neg eq(nx_j, ?)$), and depending on the value of its carried sequence number, a state with $eq(s'_i, 2)$ or $eq(s'_i, 3) \wedge sq'_i = sq_i + 1$ is reached. In the former case, node $i$ can only send a reply message, and again a state with $eq(s'_i, 0)$ is reached. In the latter case, by summand (3), node $i$ may send a request, and a state with $eq(s'_i, 4)$ is reached, while depending on $j \in ls$, node $j$ may receive such a request. If node $j$ receives such a request, then $eq(s'_j, 3) \wedge eq(src'_j, i) \wedge sq'_j = sq_j + 1$ holds. From this state, $j$ may find a path to the destination, and reply to $i$, so by summand (4), a state with $eq(s_i, 0) \wedge eq(nx_i, j) \wedge eq(sq_i, sq_j) \wedge eq(hp_i, hp_j + 1)$ can be reached. However if a node other than $j$ replies to $i$, then a state with $eq(s_i, 0) \wedge \neg eq(nx_i, j)$ is reached. If node $j$ does not receive such a request, then node $i$ may send the request again by summand (3) until it receives a reply. If $j$ never receives these requests, then states with $eq(s_i, 0) \wedge \neg eq(nx_i, j)$ are reached, but if $j$ receives one of these requests of $i$, then a state with $eq(s'_j, 3) \wedge eq(src'_j, i) \wedge sq'_j = sq_j + 1$ is reached, as already discussed.
  - By summand (3), node $j$ may receive a request from any node other than $i$ (since $\neg eq(nx_i, ?)$). With a similar argumentation as in the previous case, suppose that a state with $eq(s'_j, 3) \wedge sq'_j = sq_j + 1 \wedge \neg eq(src'_j, i)$ is reached. From this state two sets of states can be reached. Either node $i$ may not send or receive any request with a higher sequence number than its own, and consequently only states with $eq(s_i, 0) \wedge eq(nx_i, j) \wedge sq_j > sq_i$ can be reached. Or node $i$ may send or receive a request with a higher sequence number, in which case states with $eq(s_i, 0) \wedge \neg eq(nx_i, j)$ can be reached (since $\neg eq(src'_j, i)$).
  - By summand (3), both nodes $i, j$ may receive a request from a node $k$, and depending on the carried sequence number, the next states of both are 2 or 3. The first case is straightforward, as discussed in the previous cases. In the second case, a state with $eq(s'_l, 3) \wedge eq(src'_l, k) \wedge sq'_l = sq_l + 1$ where $l \in \{i, j\}$ is reached. From this state, only states with $eq(s_i, 0) \wedge \neg eq(nx_i, j)$ can be reached.

– By summand (6), node $j$ may send an error message, and depending on $i \in ls$, node $i$ may receive. If node $i$ does not receive it, then a state with $eq(s'_j, 3) \wedge sq'_j = sq_j + 1 \wedge eq(src'_j, ?)$ is reached. From this state two sets of states can be reached. Either node $i$ may not send or receive any request with a higher sequence number than its own, and consequently only states with $eq(s_i, 0) \wedge eq(nx_i, j) \wedge sq_j > sq_i$ can be reached. Or node $i$ may send or receive a request with a higher sequence number, in which case states with $eq(s_i, 0) \wedge \neg eq(nx_i, j)$ can be reached (since $eq(src'_j, ?)$). If node $i$ receives the error, a state with $eq(s'_j, 3) \wedge sq'_j = sq_j + 1 \wedge eq(src'_j, ?) \wedge eq(s'_i, 1)$ is reached. From this state, only states with $eq(s_i, 0) \wedge \neg eq(nx_i, j)$ can be reached (since $eq(src'_j, ?)$).
– By summand (6), node $i$ may send an error message, and no matter whether $j$ receives it or not, a state with $eq(s'_i, 3) \wedge sq'_i = sq_i + 1$ is reached. By summand (3), node $i$ may send a request, and as discussed before, depending on whether $j$ may receive such requests and then sends a reply to $i$, two sets of states can be reached: either $eq(s_i, 0) \wedge eq(nx_i, j) \wedge eq(sq_i, sq_i) \wedge eq(hp_i, hp_j + 1)$ or $eq(s_i, 0) \wedge \neg eq(nx_i, j)$.
– By summand (6), node $i, j$ may receive an error from a node $l$. Depending on $nx_l$, one of the nodes $i, j$ would receive it, and a state with $eq(nx'_i, 2)$ or $eq(nx'_j, 2)$ is reached. In any of these state, node $i$ or $j$ may send an error, which was discussed before.
– By summands (2) and (1), node $i, j$ may receive data from a node $l$ where $dih_l$. Depending on $nx_l$, one of the nodes $i, j$ may receive it, and a state with $\neg dih'_l \wedge dih'_i$ or $\neg dih'_l \wedge dih'_j$ is reached, as discussed before. □

**Lemma 4.** *For all $s_A$, $\ell l : LocList$ with $nodup(\ell l)$, $\xi t$, $s_B$, $src_B$, $sq_B$ and $A, B \not\in \ell l$ such that the invariants of $\mathcal{I}_{1-11}$ are satisfied, then*

$$\tau_{\widetilde{M}_2}(InitnMidDst(s_A, \ell l, \xi t, s_B, src_B, sq_B)) \simeq_b Routing(\phi(s_A, \ell l, \xi t)).$$

**Proof.** According to Proposition 1, the following conditions should be examined. Let $\eta_\tau \in I_{\widetilde{M}_1}$ where $I_{\widetilde{M}_1} = \{nsnd(\mathcal{C}, \mathfrak{m}, \ell), nrcv(\mathcal{C}, \mathfrak{m}) | \exists m \in \widetilde{M}_1 \cdot isType_m(\mathfrak{m})\}$. We use $x'$ to denote the updated value of $x$ in the next state.

1. Two cases need to be considered for the states of *InitnMidDst*: the data is held by node $A$, i.e. $eq(s_A, 0)$, or by a middle node $i \in \ell l$ (since if no node holds the data, no $\eta_\tau$ action can make $dih_j$ for some node $j$ or $eq(s_A, 0)$, and consequently the mapped state, i.e. $\neg fin$, would not change). If $eq(s_A, 0)$, then the mapped state is $eq(n, 0)$ and no $\eta_\tau$ action can change $s_A$. If $dih_i$, then the mapped state is $eq(n, hp_i)$, while $\neg fin \wedge eq(N, |\ell l|)$. The $hp_i$ may change if node $i$ receives a *rep* message by summand (4) or (5) (when $eq(s_i, 4)$). But by invariant $\mathcal{I}_5$ and $dih_i$, $eq(s_i, 0)$ holds and so it cannot receive such a message. Since an $\eta_\tau$ by another node would not change $hp_i$ and $dih_i$, the mapped state is not changed by any $\eta_\tau$.
2. Only communications of *InitnMidDst* over messages *data* are visible, which are only possible when $dih_i$ for some node $i \in \ell l$ or $eq(s_A, 0)$. Therefore three classes of states can be considered.
   - $eq(s_A, 0)$: By invariant $\mathcal{I}_4$, $\bigwedge_{i\in\ell l} \neg dih_i$ holds. In these states, for any arbitrary $lx : Loc$ and $ls : LocList$, *InitnMidDst* performs $nsnd(data(?), ?)$ for all possible topologies {} by summand (1), while node $lx$ may receive or may not receive such data (depending on $lx \in ls \wedge eq(s_{lx}, 0) \wedge \neg eq(nx_{lx}, ?)$). If node $lx$ does not receive such data, a state with $eq(s'_A, 1) \bigwedge_{i\in\ell l} \neg dih_i$ is reached, and this scenario is matched with a same action and network constraint of *Routing* which makes $eq(fin', F)$. If node $lx$ receives this message, a state with $dih'_{lx}$ is reached. This scenario can be matched with the same action and network constraint of *Routing*, by which $eq(n', hp_{lx})$.
   - $dih_i \wedge eq(nx_i, j) \wedge hp_i > 1$: By invariant $\mathcal{I}_4$, $eq(s_A, 1) \bigwedge_{j\in\ell l, \neg eq(i,j)} \neg dih_j$ holds. In these states, for any arbitrary $ls : LocList$, *InitnMidDst* performs $nsnd(data(?), ?)$ by summand (2) with the network constraint $\{? \rightsquigarrow B\}$ or {} depending on $B \in ls$, while node $j$ may receive or may not receive such data (depending on $j \in ls \wedge eq(s_j, 0) \wedge \neg eq(nx_j, ?)$). If node $j$ does not receive such a message, a state with $\neg dih'_i \wedge eq(s_A, 1) \bigwedge_{j\in\ell l, \neg eq(i,j)} \neg dih_j$ is reached, and this scenario is matched by the sending data action of *Routing* with the network constraint {} which makes $eq(fin', F)$. If the next hop (node $j$) receives, then this scenario can be matched by the sending data action of *Routing* with the network constraint {}, by which $eq(n', hp_j)$.
   - $dih_i \wedge eq(nx_i, j) \wedge eq(hp_i, 1)$: By invariant $\mathcal{I}_9$, $eq(j, B)$, and by invariant $\mathcal{I}_4$, $eq(s_A, 1) \bigwedge_{j\in\ell l, \neg eq(i,j)} \neg dih_j$. In these states, for any arbitrary $ls : LocList$, *InitnMidDst* performs $nsnd(data(B), ?)$ by summand (2) with the network constraint $\{? \rightsquigarrow B\}$ or {} depending on $B \in ls$, and a state with $\neg dih'_i \wedge eq(s_A, 1) \bigwedge_{j\in\ell l, \neg eq(i,j)} \neg dih_j$ is reached. This scenario is matched by a sending data action of *Routing* with the network constraint {} which makes $eq(fin', F)$.
3. Four cases can be considered for $\phi(s_A, \ell l, \xi t)$, i.e. the states of *Routing*:
   - *fin* : In this case no action can be performed. The same holds for *InitnMidDst*, since the mapping state is $\bigwedge_{i\in\ell l} \neg dih_i \wedge eq(s_A, 1)$, and by summands (2) and (1), data can be sent when $eq(s_A, 0)$ or $dih_i$ for some $i \in \ell l$;
   - $eq(n, 0)$: In this case *Routing* can make two $(\{\}, nsnd(data(?), ?))$ transitions, either to a state with $\neg fin'$, or for some $h < N$ to a state $eq(n', h)$. This state is mapped from states of *InitnMidDst* with $eq(s_A, 0)$. The first transition of *Routing* can be matched by the transitions of summand (1) such that the data sent by node $A$ is not received by node $lx$ $(lx \not\in ls \vee (\neg eq(s_{lx}, 0) \vee eq(nx_{lx}, ?)))$. By invariants $\mathcal{I}_{7,8}$ the hop count of each middle node is at most the number of middle nodes participating in the route discovery. Therefore, for any value of $h$, this state can do some $\eta_\tau$ actions, due to arbitrary mobility of nodes, such that for some address $lx \in \ell l$, $\neg eq(nx'_i, ?) \wedge eq(s'_i, 0) \wedge eq(hp'_i, h)$ holds, while the data is still held by $s_A$. Then this state can perform a sending data action with the network constraint {} by summand (1) for $eq(lx, i) \wedge i \in ls$ such that a state with $dih'_{lx}$ is reached. The second transition of *Routing* is matched to these data transitions.

- $n > 1 \wedge \neg fin$: Similar to the previous case, *Routing* can make two $(\{\}, nsnd(data(?), ?))$ transitions, either to a state with $\neg fin'$, or for some $h < N$ to a state with $eq(n', h)$. This state is mapped from states of *InitnMidDst* with $dih_i \wedge eq(hp_i, n) \wedge eq(nx_i, j)$ for some arbitrary address $j \in \ell l \wedge n > 0$. The first transition of *Routing* can be matched by the transitions of summand (2) such that the data sent by node $i$ is not received by node $j$ $(j \not\in ls \vee (\neg eq(s_j, 0) \vee eq(nx_j, ?)))$. By invariants $\mathcal{I}_{7,8}$ the hop count of each middle node is less than the number of middle nodes participating in the route discovery. Therefore, for any value of $h$, this state can do some $\eta_\tau$ actions, due to the arbitrary mobility changes of nodes, such a state with $\neg eq(nx_j', ?) \wedge eq(s_j', 0) \wedge eq(hp_j', h)$ is reached, while the data is still held by $i$. Then this state can perform a sending data action with the network constraint $\{\}$ by summand (2) for arbitrary $ls$ such that $j \in ls$ and a state with $dih_j'$ is reached. The second transition of *Routing* is matched to this data transition.
- $\neg fin \wedge n = 1$: In this case, *Routing* can perform $(\{\}, nsnd(data(B), ?))$, by which $fin$ is set to $F$. By invariant $\mathcal{I}_9$, the state of *InitnMidDst* implies that $eq(dih_i, T) \wedge eq(hp_i, 1) \wedge eq(nx_i, B)$ for some node $i$. By summand (2), *InitnMidDst* can perform $nsnd(data(B), ?)$ for some $ls : LocList$ such that $B \not\in ls$, while the value of $dih_i$ is set to false. $\square$

**Corollary 1.** *For all $s_A : Nat$, $\ell l : LocList$ with $nodup(\ell l)$, $\xi t$, $s_B$, $src_B$, $sq_B$ and $A, B \not\in \ell l$ such that the invariants of $\mathcal{I}_{1-11}$ are satisfied,*

$$(\{\}, \tau).Routing(\phi(s_A, \ell l, \xi t)) \simeq_{rb} (\{\}, \tau).\tau_{\widetilde{M}_2}(\partial_{\widetilde{M}_1}((\nu A) \\ [\![Init(s_A, A, B)]\!]_A \parallel (\nu \ell l) nMid(\ell l, \xi t) \parallel [\![Dst(s_B, src_B, sq_B, B)]\!]_B)).$$

Eq. (4) is a direct result of following corollary.

**Theorem 2.** *For all $\ell l : LocList$ with $nodup(\ell l)$, $\xi t$ such that for all $i \le |\ell l|$ the $i^{th}$ entry of table $\xi t$ holds $\langle 0, ?, ?, 0, 0, i, F\rangle$:*

$$(\{\}, \tau).Routing(0, |\ell l|, F) \simeq_{rb} (\{\}, \tau).\tau_{\widetilde{M}_2}(\partial_{\widetilde{M}_1}((\nu A) \\ [\![Init(0, A, B)]\!]_A \parallel (\nu \ell l) nMid(\ell l, \xi t) \parallel [\![Dst(0, ?, 0, B)]\!]_B)).$$

## 6. Conclusions and future work

In this paper, we enhanced and illustrated the applicability of our framework *CNT*, tailored for the specification and verification of MANETs. To this aim, we examined the applicability of the *CNT* operational semantics, constrained labeled transition systems, in model checking. Through model checking we can examine the behavior of a MANET for the arbitrary mobility of nodes through one model, without the need to specify mobility changes. The constraints added to the transition labels allow us to derive mobility scenarios for each MANET behavior. Then we extended our framework with symbolic verification technique based on cones and foci, and demonstrated its application to the verification of MANETs with an arbitrary number of nodes which deploy the same protocol. We aim to establish a framework for mechanical protocol verification following the approach of [8]. Our algebraic framework is the first one that addresses the verification of networks with an arbitrary number of nodes. In [23] an approach using a model checker (*SPIN*) and a theorem prover (*HOL*) was presented to reason about such networks; the theorem prover uses the facts proved by model checker. However, breaking down a proof to these facts is not straightforward. The symbolic verification approach provides a more natural proof framework for such networks.

Our framework is applicable in wireless networks in which communication is based on non-blocking and lossy local broadcast, if it is extended with the static location binding operator of [24] which restricts the arbitrary mobility of nodes.

## Appendix. Proof of Proposition 1

We exploit semi-branching computed network bisimilarity introduced in [3] to prove Proposition 1. In the next definition, $t \xrightarrow{[(\mathcal{C}, \eta)]} t'$ denotes either $t \xrightarrow{(\mathcal{C}, \eta)} t'$, or $t = t'$ if $\eta$ is of the form $nrcv(\mathfrak{m})$ or $\tau$.

**Remark 1.** As axiom $Ch_6$ explains, if $t \xrightarrow{(\mathcal{C}_1, \eta)} t'$, then $t \xrightarrow{(\mathcal{C}_2, \eta)} t'$ for any $\mathcal{C}_1 \subseteq \mathcal{C}_2$.

**Definition 6.** A binary relation $\mathcal{R}$ on computed network terms is a semi-branching computed network simulation, if $t_1 \mathcal{R} t_2$ implies whenever $t_1 \xrightarrow{(\mathcal{C}, \eta)} t_1'$:

- there are $t_2'$ and $t_2''$ such that $t_2 \Rightarrow t_2'' \xrightarrow{[(\langle \mathcal{C}, \eta)\rangle]} t_2'$, $t_1 \mathcal{R} t_2''$ and $t_1' \mathcal{R} t_2'$.

$\mathcal{R}$ is a semi-branching computed network bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are semi-branching computed network simulations. Computed networks $t_1$ and $t_2$ are semi-branching computed network bisimilar if $t_1 \mathcal{R} t_2$, for some semi-branching computed network bisimulation relation $\mathcal{R}$.

**Theorem 3.** *Two computed network terms are related by a branching computed network bisimulation if and only if they are related by a semi-branching computed network bisimulation [3].*

To prove Proposition 1, in view of Theorem 3, instead of showing that the state mapping relation $\phi : D \to D'$ constitutes a branching computed network bisimulation, we show that it constitutes a semi-branching computed network bisimulation on the reachable states of $D$, overapproximated by the invariant $\mathcal{I}$. We assume without loss of generality that $D$ and $D'$ are disjoint. Define $\mathcal{R} \subseteq D \times D'$ as the smallest relation such that whenever $\mathcal{I}(d)$ for a $d : D$, then $d\mathcal{R}\phi(d)$. Then we show that $\mathcal{R}$ satisfies the transfer conditions of Definition 6. Let $s\mathcal{R}t$ such that $t = \phi(s)$. By definition of $\mathcal{R}$ we have $\mathcal{I}(s)$.

- If $s \xrightarrow{(\mathcal{C},\eta)} s'$, there are two cases to consider:
  1. If $\eta = \tau$, then it must be generated by application of the abstraction function $\tau_{\widetilde{M}}$ on an action $\eta_\tau \in I_{\widetilde{M}}$, while $h_{\eta_\tau}(s, e)$, $s' = g_{\eta_\tau}(s, e)$ and $\mathcal{C} = \mathcal{C}_{\eta_\tau}(s, e)$ for some $e : E$. By matching criterion 1, $\phi(g_{\eta_\tau}(s, e)) = t$. Moreover, $\mathcal{I}(s)$ and $h_{\eta_\tau}(s, e)$ together imply that $\mathcal{I}(g_{\eta_\tau}(s, e))$. Hence, by definition of $\mathcal{R}$, $g_{\eta_\tau}(s, e)\mathcal{R}t$.
  2. If $\eta \neq \tau$, then $h_\eta(s, e), s' = g_\eta(s, e)$ and $\mathcal{C} = \mathcal{C}_\eta(s, e)$ for some $\eta \in Act^c \backslash I_{\widetilde{M}}$ and $e : E$. By matching criterion 2, either $\eta$ is a receive action such that $\phi(g_\eta(s, e)) = t$, or there is an $\langle\eta\rangle$ such that $h'_{\langle\eta\rangle}(t, e), f_\eta(s, e) = f'_{\langle\eta\rangle}(t, e), \mathcal{C}'_{\langle\eta\rangle}(t, e) \subseteq \mathcal{C}_\eta(s, e)$ and $\phi(g_\eta(s, e)) = g'_{\langle\eta\rangle}(t, e)$. Moreover, $\mathcal{I}(s)$ and $h_\eta(s, e)$ together imply $\mathcal{I}(g_\eta(s, e))$. In the former case, by definition of $\mathcal{R}$, $g_\eta(s, e)\mathcal{R}t$. In the latter case, by Remark 1 and $\langle(\mathcal{C}_\eta(s, e), \eta)\rangle = (\mathcal{C}_{\langle\eta\rangle}(s, e), \langle\eta\rangle)$, $t \xrightarrow{\langle(\mathcal{C}_\eta(s,e),\eta)\rangle} g'_{\langle\eta\rangle}(t, e)$ and consequently $g_\eta(s, e)\mathcal{R}g'_{\langle\eta\rangle}(t, e)$.
- If $t \xrightarrow{\eta}_{\mathcal{C}} t'$, then $h'_\eta(t, e)$, $t' = g'_\eta(t, e)$ and $\mathcal{C} = \mathcal{C}'_\eta(t, e)$ for some $\eta \in Act^c \setminus I_{\widetilde{M}}$ and $e : E$. By matching criterion 3, either $\eta$ is a receive action such that $t = t'$, or there is an $s^* : D$ such that $s \xrightarrow{\eta_{\tau_1}}_{\mathcal{C}_1} \ldots \xrightarrow{\eta_{\tau_n}}_{\mathcal{C}_n} s^*$ with $\eta_{\tau_1}, \ldots, \eta_{\tau_n} \in I_{\widetilde{M}}$ and $h_{\langle\eta\rangle}(s^*, e), f_{\langle\eta\rangle}(s^*, e) = f'_\eta(t, e)$, $\mathcal{C}_{\langle\eta\rangle}(s^*, e) \subseteq \mathcal{C}'_\eta(t, e)$ and $\phi(g_{\langle\eta\rangle}(s^*, e)) = g'_\eta(t, e)$ in the *CLTS* for *Imp*. Invariant $\mathcal{I}$ and matching criterion 1 hold for all states on this $\eta_\tau$-path. Repeatedly applying matching criterion 1, we get $\phi(s^*) = \phi(s) = t$. The former case is straightforward since $\mathcal{I}(s^*)$, and by definition of $\mathcal{R}$, $s^*\mathcal{R}t$. In the latter case by Remark 1 and $\langle(\mathcal{C}'_\eta(t, e), \eta)\rangle = (\mathcal{C}'_{\langle\eta\rangle}(t, e), \langle\eta\rangle)$, $s \Rightarrow s^* \xrightarrow{\langle(\mathcal{C}'_\eta(t,e),\eta)\rangle} g_{\langle\eta\rangle}(s^*, e)$. Moreover, $\mathcal{I}(s^*)$ and $h_{\langle\eta\rangle}(s^*, e)$ together imply $\mathcal{I}(g_{\langle\eta\rangle}(s^*, e))$. So by definition of $\mathcal{R}$, $s^*\mathcal{R}t$ and $g_{\langle\eta\rangle}(s^*, e)\mathcal{R}g'_\eta(t, e)$.

Concluding, $\mathcal{R}$ is a semi-branching computed network bisimulation.

## References


[1] F. Ghassemi, W. Fokkink, A. Movaghar, Restricted broadcast process theory, in: Proc. 6th Conference on Software Engineering and Formal Methods, SEFM'08, IEEE, 2008, pp. 345–354.
[2] F. Ghassemi, W. Fokkink, A. Movaghar, Equational reasoning on ad hoc networks, in: Proc. 3rd Conference on Fundamentals of Software Engineering, FSEN'09, in: Lecture Notes in Computer Science, vol. 5961, Springer, 2009, pp. 113–128.
[3] F. Ghassemi, W. Fokkink, A. Movaghar, Equational reasoning on mobile ad hoc networks, Fundamenta Informaticae 103 (2010) 1–41.
[4] J.F. Groote, A. Mathijssen, M. Reniers, Y. Usenko, M. van Weerdenburg, The formal specification language mCRL2, in: Proc. Methods for Modelling Software Systems, MMOSS'06, in: Dagstuhl Seminar Proceedings, vol. 06351, Schloss Dagstuhl, 2006.
[5] H. Garavel, R. Mateescu, F. Lang, W. Serwe, CADP 2006: A toolbox for the construction and analysis of distributed processes, in: Proc. 19th Conference on Computer Aided Verification, CAV'07, in: Lecture Notes in Computer Science, vol. 4590, Springer, 2007, pp. 158–163.
[6] J.F. Groote, J. Springintveld, Focus points and convergent process operators: A proof strategy for protocol verification, Journal of Logic and Algebraic Programming 49 (1–2) (2001) 31–60.
[7] W. Fokkink, J. Pang, Cones and foci for protocol verification revisited, in: Proc. 6th Conference on Foundations of Software Science and Computational Structures, FoSSaCS'06, in: Lecture Notes in Computer Science, vol. 2620, Springer, 2003, pp. 267–281.
[8] W. Fokkink, J. Pang, J. van de Pol, Cones and foci: A mechanical framework for protocol verification, Formal Methods in System Design 29 (1) (2006) 1–31.
[9] J.F. Groote, J.v. Wamel, The parallel composition of uniform processes with data, Theoretical Computer Science 266 (1–2) (2001) 631–652.
[10] W. Fokkink, Modelling Distributed Systems, Springer, 2007.
[11] H. Ehrich, J. Loeckx, M. Wolf, Specification of Abstract Data Types, John Wiley, 1996.
[12] J.F. Groote, A. Ponse, $\mu$CRL: a base for analysing processes with data, in: Proc. 3rd Workshop on Concurrency and Compositionality, in: GMD-Studien, vol. 191, 1991, pp. 125–130.
[13] J.F. Groote, A. Ponse, Syntax and semantics of $\mu$CRL, in: Workshop on Algebra of Communicating Processes, in: Workshops in Computing, Springer, 1995, pp. 26–62.
[14] F. Ghassemi, W. Fokkink, A. Movaghar, Verification of mobile ad hoc networks: an algebraic approach, Tech. Rep., Computer Engineering Department, Sharif University of Technology.
[15] J. Bergstra, J.W. Klop, Process algebra for synchronous communication, Information and Control 60 (1–3) (1984) 109–137.
[16] B. Luttik, Choice quantification in process algebra, Ph.D. Thesis, University of Amsterdam, 2002.
[17] C.E. Perkins, E.M. Belding-Royer, Ad-hoc on-demand distance vector routing, in: Proc. 2nd Workshop on Mobile Computing Systems and Applications, WMCSA'99, IEEE, 1999, pp. 90–100.
[18] O. Wibling, J. Parrow, A. Pears, Automatized verification of ad hoc routing protocols, in: Proc. 24th IFIP WG6.1 International Conference on Formal Techniques for Networked and Distributed Systems, FORTE'04, in: Lecture Notes in Computer Science, vol. 3235, Springer, 2004, pp. 343–358.
[19] R. Mateescu, M. Sighireanu, Efficient on-the-fly model-checking for regular alternation-free mu-calculus, Science of Computer Programming 46 (3) (2003) 255–281.
[20] D. Obradovic, Formal analysis of routing protocols, Ph.D. Thesis, University of Pennsylvania, 2001.
[21] Y. Usenko, Linearization in $\mu$CRL, Ph.D. Thesis, Eindhoven University of Technology, 2002.
[22] M. Bezem, J.F. Groote, Invariants in process algebra with data, in: Proc. 5th International Conference of Concurrency Theory, CONCUR'94, in: Lecture Notes in Computer Science, vol. 836, Springer, 1994, pp. 401–416.
[23] K. Bhargavan, D. Obradovic, C.A. Gunter, Formal verification of standards for distance vector routing protocols, Journal of the ACM 49 (4) (2002) 538–576.
[24] J.C. Godskesen, A Calculus for mobile ad-hoc networks with static location binding, Electronic Notes in Theoretical Computer Science 242 (1) (2009) 161–183.