

# A process-calculus analysis of a MANET algorithm

Dimitrios Kouzapas<sup>1</sup> and Anna Philippou<sup>2</sup>

<sup>1</sup> Imperial College, London, UK. dk208@doc.ic.ac.uk

<sup>2</sup> University of Cyprus, Cyprus. annap@cs.ucy.ac.cy

**Abstract.** In this paper we propose a process-calculus framework for mobile ad hoc networks (MANETs). The proposed calculus allows one to abstract away from neighbor-discovery computations and contains features for viewing networks at different levels of abstraction. We develop a theory of confluence for the calculus and we use the machinery developed towards the specification and analysis of a leader-election algorithm for MANETs.

## 1 Introduction

Distributed and wireless systems present today one of the most challenging areas of research in computer science. Their high complexity and dynamic nature and features such as broadcasting communication, mobility and fault tolerance, render their construction, description and analysis a challenging task. The development of formal frameworks for describing and associated methodologies for reasoning about distributed systems has been an active area of research for the last few decades. Process calculi, such as CCS [8] and  $\pi$ -calculus [9], are one such formalism. Since their inception, they have been extensively studied and they have proved quite successful in the modeling and reasoning about systems. They have been extended for modeling a variety of aspects of process behavior including mobility, distribution and broadcasting.

Our goal in this paper is to propose a process calculus in which to be able to reason about mobile ad hoc networks (MANETs) and their protocols. Our proposal combines ideas from works which have previously appeared in the literature [14, 2, 6, 7] on issues such as broadcasting, movement and separating between a node's control and topology information. However, we have found the need of departing from these works due to the difficulty we experienced while considering specific case studies. Thus, our calculus extends existing proposals in the following ways.

1. To begin with our protocol allows nodes to broadcast messages at different transmission levels. This is a common feature of ad hoc networks also present in the leader-election algorithm we consider, where a leader of a component may send heartbeat messages to a wider range of nodes than it would typically broadcast to.
2. Second, we have found it convenient to extend the semantics of our calculus with rules implementing a neighbor-discovery protocol. Knowledge of the precise set of neighbors of ad hoc network nodes is a key issue to protocol correctness. To take this into account, one might model an actual neighbor-discovery protocol in parallel to the algorithm under study and verify the composition of the two. Although such a study would be beneficial towards obtaining a better understanding on the behavior of both protocols, it would turn an already laborious task into a more laborious one

and it would impose further requirements on behalf of the modeling language (e.g. to reason about timed behaviors).

3. Lastly, we introduce a hiding construct in our language that allows us to observe networks at different levels of abstraction. Since messages in our network descriptions are *broadcasted* over the medium, the notion of channel restriction (or name hiding) becomes irrelevant. Nonetheless, the effect of hiding behaviors remains useful in our setting, especially for analysis purposes. To achieve this, we associate every message with a type and we implement hiding by restricting the set of message types which should be observable at the highest level of a process.

The operational semantics of our calculus is given in terms of a labelled transition system on which we propose a notion of weak bisimulation for the language. Subsequently, we develop a theory of confluence. The notion of confluence was first studied in the context of concurrent systems by Milner in CCS [8] and subsequently in various other settings [15, 3, 5, 12, 10, 13, 11]. Its essence, is that “of any two possible actions, the occurrence of one will never preclude the other”. As shown in the mentioned papers, confluence implies determinacy and semantic-invariance under internal computation, and it is preserved by several system-building operators. These facts make it possible to reason compositionally that a system is confluent and to exploit this fact while reasoning about its behavior. This paper, is the first to consider the theory of confluence in a setting of broadcasting communication. We establish a variety of results including a theorem that allows for compositional reasoning of confluent behavior. We illustrate the utility of these techniques as well as the formalism via the specification and the verification of a leader-election algorithm.

The remainder of the paper is structured as follows. In the next section we introduce the syntax and semantics of our calculus, and develop its theory of bisimulation and confluence. Section 3 contains an application of our methodology for establishing the correctness of a MANET leader-election algorithm and Section 4 concludes the paper. Due to the page limitations, the proofs of Sections 2.3 and 3.3 either are missing or only sketched within this extended abstract. The complete proofs can be found in [4].

## 2 The Process Calculus

In our calculus, CSDT (Calculus for Systems with Dynamic Topology), we consider a system as a set of nodes operating in space and communicating with each other by broadcasting messages. A node possesses a location and a unique identifier. Messages broadcasted by a node can be received by nodes within its transmission range. We consider processes to be able to broadcast messages at different levels, a normal level, typically used for communication and a high level, used for special messages when this is deemed necessary by a protocol. Each of these modes is associated with a distinct transmission range which we assume to be the same for all nodes.

Based on this we consider a process to be able to participate in three types of actions: (1) a normal-level broadcast of a message, (2) a high-level broadcast, or a heartbeat, of a message, or (3) a receipt of a message. A broadcast message is received by all nodes which fall within the normal transmission range of the transmitting node and a heartbeat message is received by all nodes which fall within the high transmission range of the transmitting node.

Movement in our calculus is modeled as the spontaneous change in the location of a node, with the restriction that the originating and the destination locations are neighboring locations. The notion of neighborhood is implemented via a relation  $N$  where  $(\ell, \ell') \in N$  exactly when locations  $N$  and  $N'$  are neighbors.

Finally, we embed in our calculus the implementation of a protocol that allows nodes to observe changes in the network's topology and, specifically, to observe the departure or arrival of nodes within their normal and high transmission ranges. This addition is implemented via four rules in our semantics.

## 2.1 The Syntax

We now continue to formalize the above intuitions into the syntax of CSDT. We begin by describing the basic entities of the calculus. We assume a set of locations  $L$  ranged over by  $\ell, \ell'$  and a function specifying the distance between pairs of locations  $d : L \times L \times Real$ . Messages transmitted by a node at location  $\ell$  are received by a process at location  $\ell'$  when  $d(\ell, \ell') \leq r_n$ , if the message was sent in the normal mode, and when  $d(\ell, \ell') \leq r_h$ , if the message was sent in the high mode, where  $r_n$  and  $r_h$  are the respective transmission ranges of nodes, assumed to be the same for all nodes. Furthermore, we assume a set of special labels  $T$  which are associated with messages during their broadcast over the medium, ranged over by  $t, \mathbf{ack}, \mathbf{leader}$ , etc.

Further, we assume a set of *terms*, ranged over by  $e$ , built over (1) a set of *constants*, ranged over by  $u, v$ , (2) a set of *variables* ranged over by  $x, y$ , and (3) function applications of the form  $f(e_1, \dots, e_n)$  where  $f$  is a function from a set of functions, and the  $e_i$  are terms. We say that a term is *closed* if it contains no variables. The evaluation relation  $\rightarrow$  for closed terms is defined in the expected manner. We write  $\tilde{r}$  for a tuple of syntactic entities  $r_1, \dots, r_n$ .

Finally, we assume a set of process constants  $C$ , denoted by  $C$ , each with an associated definition of the form  $C\langle\tilde{x}\rangle \stackrel{\text{def}}{=} P$ , where the node  $P$  may contain occurrences of  $C$ , as well as other constants. The syntax of our calculus is given by the following BNF definition, where  $T \subseteq T$ :

$$\begin{aligned}
P &::= \mathbf{0} \mid \alpha.P \mid P_1 + P_2 \mid \text{cond}(e_1 \triangleright P_1, \dots, e_n \triangleright P_n) \mid C\langle\tilde{v}\rangle \\
\alpha &::= \bar{b}(w, t, \tilde{v}) \mid \bar{h}(w, t, \tilde{v}) \mid r(t, \tilde{x}) \mid \tau \\
b &::= - \mid i \\
N &::= \mathbf{0} \mid P\sigma \mid N_1 \mid N_2 \mid N \setminus T \\
\sigma &::= \llbracket id, \ell, N_i, H_i \rrbracket
\end{aligned}$$

According to the definition,  $P$  ranges over processes and represents the code run by a node of a network. The  $\mathbf{0}$  process represents the inactive process.  $\alpha.P$  describes the process which first engages in action  $\alpha$  and then behaves as  $P$ . Action  $\alpha$  can be a broadcast, a heartbeat, a receive or an internal action. A broadcast action  $\bar{b}(w, t, \tilde{v})$  (similarly, a heartbeat action  $\bar{h}(w, t, \tilde{v})$ ) is a transmission of type  $t$  of the tuple  $\tilde{v}$  with intended recipients  $b$ , where  $'-'$  denotes that the message is intended for all neighbors of the transmitting node and  $j$  denotes that it is intended solely for node  $j$ .  $P_1 + P_2$  represents the nondeterministic choice between  $P_1$  and  $P_2$ . Process constants provide a mechanism for including recursion in the calculus. Finally, the conditional process

$\text{cond } (e_1 \triangleright P_1, \dots, e_n \triangleright P_n)$  presents the conditional choice between a set of processes: it behaves as  $P_i$ , where  $i$  is the smallest integer for which  $e_i$  evaluates to true.

On the other hand, networks are built on the basis of located processes,  $P\sigma$ , where  $\sigma$  is the node's topology information which we call its interface. An interface  $\sigma$  contains the node identifier  $id$ , its location  $\ell$  as well as its normal and heartbeat neighbors  $N_i$  and  $H_i$ , according to its current knowledge. Note that we allow the control part of a located process  $P\sigma$ , namely  $P$ , to refer to the information mentioned in the interface  $\sigma$ , by using the special labels  $id$ ,  $l$ ,  $N_i$  and  $H_i$ , thus allowing the control part of a process to express dependencies on the node's neighborhood information. For example, an expression “ $4 \in N_i$ ” occurring within  $P[[1, \ell, \{2\}, \{2, 3\}]]$  is evaluated as “ $4 \in \{2\}$ ”.

Thus, a network can be an inactive network  $\mathbf{0}$ , a located node  $P\sigma$ , a parallel composition of networks, or a restricted network  $P \setminus T$ . The restricted network  $P \setminus T$ , where  $T \subseteq \mathbb{T}$ , restricts the scope of messages of all types  $t \in T$ , to network  $N$ : components of  $N$  may exchange messages of these types to interact with one another but not with  $N$ 's environment. In what follows, given an interface  $\sigma$ , we write  $l(\sigma)$  and  $id(\sigma)$  for the location and the identifier mentioned in  $\sigma$ , respectively.

## 2.2 The Semantics

The semantics of the calculus is defined in terms of a structural congruence relation  $\equiv$  (see Table 3 in the Appendix) and a structural operational semantics. The latter is given at two levels in Tables 1 and 2. The rules of Table 1 describe the behavior of located processes in isolation whereas the rules in Table 2 prescribe the behavior of a network. In both cases a transition of  $P$  (or  $N$ ) has the form  $P \xrightarrow{\alpha} P'$ , specifying that  $P$  can perform action  $\alpha$  and evolve into  $P'$ .

With respect to Table 1, it is worth noting that in actions of sending and receiving messages, we embed the location of the process participating in the communication (both for sending and receiving) and the identifier of the receiving agent, in the case of receiving. This is needed for implementing communication within a network as described in the rules of Table 2.

**Table 1. Transition rules for located nodes**

(BCast) $\bar{b}(w, t, \tilde{v}).P\sigma \xrightarrow{\bar{b}(w, l(\sigma), t, \tilde{v})} P\sigma$	(HBeat) $\bar{h}(w, t, \tilde{v}).P\sigma \xrightarrow{\bar{h}(w, l(\sigma), t, \tilde{v})} P\sigma$
(Rec) $r(t, \tilde{x}).P\sigma \xrightarrow{r(id(\sigma), l(\sigma), t, \tilde{v})} P\{\tilde{v}/\tilde{x}\}\sigma$	(Sum) $\frac{P_i\sigma \xrightarrow{\alpha} P'_i\sigma, i \in \{1, 2\}}{(P_1 + P_2)\sigma \xrightarrow{\alpha} P'_i\sigma}$
(Cond) $\frac{P_i\sigma \xrightarrow{\alpha} P'_i\sigma}{(\text{cond } (e_1 \triangleright P_1, \dots, e_n \triangleright P_n))\sigma \xrightarrow{\alpha} P'_i\sigma} \quad e_i \rightarrow \text{true}, \forall j < i, e_j \rightarrow \text{false}$	
(Move) $\frac{(\ell, \ell') \in N}{P[[i, \ell, N_i, H_i]] \xrightarrow{\tau} P[[i, \ell', N_i, H_i]]}$	(Const) $\frac{[P\{\tilde{v}/\tilde{x}\}]\sigma \xrightarrow{\alpha} P'\sigma}{[C(\tilde{v})]\sigma \xrightarrow{\alpha} P'\sigma} \quad C(\tilde{x}) \stackrel{\text{def}}{=} P$

Moving our attention to the rules of Table 2, we point out that the first four rules implement the underlying neighbor discovery protocol. Nodes which are within the normal and high transmission ranges of a located process are included in the appropriate sets in its interface and, similarly, nodes that have exited these transmission ranges are removed from the appropriate sets in its interface. The three (Broadcast) rules give semantics to broadcasting within the language. If the broadcast has an intended recipient and this recipient is within the range of the broadcast, the message is received and the action is transformed into an internal action (Broadcast1). If the broadcast is intended for all neighbors of the sender and a recipient is available within the transmission range, then the message is received but the broadcast remains in the medium to be caught by other available receivers (Broadcast2). Finally, if a broadcast is available but there is no appropriate receiver then, again, the broadcast remains in the medium (Broadcast3). The rules for heartbeat communication are similar. Moving on to the rules for hiding, we observe that the effect of hiding is to transform all actions involving messages of the restricted set  $T$  into internal actions.

### 2.3 Bisimulation and Confluence

In this section we build some machinery for reasoning about networks described in CSDP. First, let us recall that  $Q$  is a *derivative* of  $P$ , if there exist actions  $\alpha_1, \dots, \alpha_n$ ,  $n \geq 0$ , such that  $P \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_n} Q$ . Moreover, given an action  $\alpha$  we write  $P \Longrightarrow Q$  for  $P \xrightarrow{(\tau)}^* Q$ ,  $P \xrightarrow{\alpha} Q$  for  $P \xrightarrow{\alpha} \Longrightarrow Q$ , and  $P \xrightarrow{\hat{\alpha}} Q$  for  $P \Longrightarrow Q$  if  $\alpha = \tau$  and  $P \xrightarrow{\alpha} Q$  otherwise.

The first useful tool which accompanies our calculus is a notion of observational equivalence. Below we introduce the relation on which we base our study.

**Definition 1.** *Bisimilarity* is the largest symmetric relation, denoted by  $\approx$ , such that, if  $P \approx Q$  and  $P \xrightarrow{\alpha} P'$ , there exists  $Q'$  such that  $Q \xrightarrow{\hat{\alpha}} Q'$  and  $P' \approx Q'$ .

Another concept used in our study is the notion of *confluence*. We begin with the notion of determinacy which may be expressed as follows:

**Definition 2.**  $N$  is *determinate* if, for every derivative  $M$  of  $N$  and for all actions  $\alpha$ , whenever  $M \xrightarrow{\alpha} M'$  and  $M \xrightarrow{\hat{\alpha}} M''$  then  $M' \approx M''$ .

This definition makes precise the requirement that, when an experiment is conducted on a process, it should always lead to the same state up to bisimulation. Determinacy has been extended into the notion of *confluence* as follows:

**Definition 3.**  $N$  is *confluent* if it is determinate and, for each of its derivatives  $M$  and distinct actions  $\alpha, \beta$ , if  $M \xrightarrow{\alpha} M_1$  and  $M \xrightarrow{\beta} M_2$  then, there are  $M'_1$  and  $M'_2$  such that  $M_2 \xrightarrow{\hat{\alpha}} M'_2$ ,  $M_1 \xrightarrow{\hat{\beta}} M'_1$  and  $M'_1 \approx M'_2$ .

We have shown that confluence implies semantic-invariance under internal computation, and it is preserved by several system-building operators of our calculus. These facts make it possible to reason compositionally that a system is confluent and to exploit this fact while reasoning about its behavior.

**Table 2. Transition rules for networks**

(InScopeN)	$\frac{d(\ell, l(\sigma)) \leq r_n, \text{id}(\sigma) \notin N_i}{P[[id, \ell, N_i, H_i] \mid Q\sigma \xrightarrow{\tau} P[[id, \ell, N_i \cup \{\text{id}(\sigma)\}, H_i] \mid Q\sigma}}$
(InScopeH)	$\frac{d(\ell, l(\sigma)) \leq r_h, \text{id}(\sigma) \notin H_i}{P[[id, \ell, N_i, H_i] \mid Q\sigma \xrightarrow{\tau} P[[id, \ell, N_i, H_i \cup \{\text{id}(\sigma)\}] \mid Q\sigma}}$
(OutofScopeN)	$\frac{d(\ell, l(\sigma)) > r_n, \text{id}(\sigma) \in N_i}{P[[id, \ell, N_i, H_i] \mid Q\sigma \xrightarrow{\tau} P[[id, \ell, N_i - \{\text{id}(\sigma)\}, H_i] \mid Q\sigma}}$
(OutofScopeH)	$\frac{d(\ell, l(\sigma)) > r_h, \text{id}(\sigma) \in H_i}{P[[id, \ell, N_i, H_i] \mid Q\sigma \xrightarrow{\tau} P[[id, \ell, N_i, H_i - \{\text{id}(\sigma)\}] \mid Q\sigma}}$
(Broadcast1)	$\frac{N_1 \xrightarrow{\bar{b}(i, \ell, t, \bar{v})} N'_1, N_2 \xrightarrow{r(i, \ell', t, \bar{v})} N'_2, d(\ell, \ell') \leq r_n}{N_1 \mid N_2 \xrightarrow{\tau} N'_1 \mid N'_2}$
(Broadcast2)	$\frac{N_1 \xrightarrow{\bar{b}(-, \ell, t, \bar{v})} N'_1, N_2 \xrightarrow{r(id, \ell', t, \bar{v})} N'_2, d(\ell, \ell') \leq r_n}{N_1 \mid N_2 \xrightarrow{\bar{b}(-, \ell, t, \bar{v})} N'_1 \mid N'_2}$
(Broadcast3)	$\frac{N_1 \xrightarrow{\bar{b}(w, \ell, t, \bar{v})} N'_1 \text{ and not } N_2 \xrightarrow{r(i, \ell', t, \bar{v})} N'_2 \text{ with } i = w, \text{ or } d(\ell, \ell') \leq r_n}{N_1 \mid N_2 \xrightarrow{\bar{b}(w, \ell, t, \bar{v})} N'_1 \mid N'_2}$
(HeartBeat1)	$\frac{N_1 \xrightarrow{\bar{h}(i, \ell, t, \bar{v})} N'_1, N_2 \xrightarrow{r(i, \ell', t, \bar{v})} N'_2, d(\ell, \ell') \leq r_h}{N_1 \mid N_2 \xrightarrow{\tau} N'_1 \mid N'_2}$
(HeartBeat2)	$\frac{N_1 \xrightarrow{\bar{h}(-, \ell, t, \bar{v})} N'_1, N_2 \xrightarrow{r(id, \ell', t, \bar{v})} N'_2, d(\ell, \ell') \leq r_h}{N_1 \mid N_2 \xrightarrow{\bar{h}(-, \ell, t, \bar{v})} N'_1 \mid N'_2}$
(HeartBeat3)	$\frac{N_1 \xrightarrow{\bar{h}(w, \ell, t, \bar{v})} N'_1 \text{ and not } N_2 \xrightarrow{r(i, \ell', t, \bar{v})} N'_2 \text{ with } i = w, \text{ or } d(\ell, \ell') \leq r_h}{N_1 \mid N_2 \xrightarrow{\bar{h}(w, \ell, t, \bar{v})} N'_1 \mid N'_2}$
(Hide1)	$\frac{N \xrightarrow{\alpha} N' \text{ and } \text{type}(\alpha) \notin T}{N \setminus T \xrightarrow{\alpha} N' \setminus T}$
(Hide2)	$\frac{N \xrightarrow{\alpha} N' \text{ and } \text{type}(\alpha) \in T}{N \setminus T \xrightarrow{\tau} N' \setminus T}$
(Par)	$\frac{N_1 \xrightarrow{\tau} N'_1}{N_1 \mid N_2 \xrightarrow{\tau} N'_1 \mid N'_2}$
(Struct)	$\frac{N \equiv M, M \xrightarrow{\alpha} M', M' \equiv N'}{N \xrightarrow{\alpha} N'}$

In this work, we will employ the following additional notion and result for aiding the verification process. These are inspired by similar results proposed in [13] in the context of the  $\pi$ -calculus but adopted for our framework.

**Definition 4.**  $N$  is *o-determinate* if, for every derivative  $M$  of  $N$  and for  $c \in \{b, h\}$ , whenever  $M \xrightarrow{\bar{c}(w, \ell, t, \tilde{v})} M'$  and  $M \xrightarrow{\bar{c}(w, \ell, t, \tilde{u})} M''$ , then  $\tilde{v} = \tilde{u}$  and  $M' \approx M''$ .

**Theorem 1.** Suppose  $N = (N_1 \mid \dots \mid N_k) \setminus T$ , where each  $N_j$  is confluent and *o-determinate*. Then  $N$  is confluent.

### 3 Specification and Verification of a Leader-Election Algorithm

#### 3.1 The algorithm

The algorithm we consider for our case study is the distributed leader-election algorithm presented in [16]. It operates on an arbitrary topology of nodes with distinct identifiers and it elects as the leader of the network the node with the maximum identifier.

We first describe the static version of the algorithm which assumes that no topology changes are possible. We then proceed to extend this description to the mobile setting.

In brief, the static algorithm operates as follows. In its initial state, a network node may initiate a leader-election computation (note that more than one node may do this) or accept leader-election requests from its neighbors. Once a node initiates a computation, it triggers communication between the network nodes which results into the creation of a spanning tree of the graph: each node picks as its father the node from which it received the first request and forwards the request to all of its remaining neighbors. Each node awaits to receive from each of its children the maximum identifier of the subtree at which they are rooted and, then, forward it to its father. Naturally, the root will receive the maximum identifier of the entire computation tree which is the elected leader. The leader is then flooded to the entire network.

Note that if more than one node initiates a leader-election computation then only one computation survives. This is established by associating each computation with a source identifier. Whenever a node already in a computation receives a request for a computation with a greater source, it abandons its original computation and it restarts a computation with this new identifier.

In the mobile setting, it is easy to observe that with node mobility, crashes and failures as well as network partitions and merges, the above algorithm is inadequate. To begin with, let us note that once a leader is elected it emits heartbeat messages to other nodes in its vicinity. The absence of a heartbeat message from its leader triggers a node to initiate a new computation of a leader. In this case computations proceeds as described by the static algorithm with the exception of the following fine points:

- **Losing a child node.** If a node loses a child then it removes the child from the set of nodes from which it expects an acknowledgement and continues its computation.
- **Losing a parent node.** If a node loses its father then it assigns itself the role of the root of its subtree and continues its computation.

- **Partition Merges** If two components of the system merge, they each proceed with their computation (if they are still computing a leader) ignoring any invitations to join the other’s computation. Once they elect their individual leaders and start flooding their results, they both adopt the leader with the smallest identifier.

### 3.2 Specification of the Protocol

In this subsection we describe the algorithm in CSDT. We assume that messages exchanged by the nodes of the network may be of one of the following types:

- **election**: message of such a type are invitations by a node to another to join its computation.
- **ack1**: a message of such a type notifies the recipient that the sender has agreed to enter its computation and commits to forward the maximum identifier among its downward nodes.
- **ack0**: a message of a such a type notifies the recipient that the sender has not agreed to be one of its children.
- **leader**: a message of this type announces the elected leader of a computation during the flooding process.
- **reply**: a message of this type announces the computation in which a node participates. Note that such messages are important for the following reason: If a node  $x$  departs from a neighborhood and enters a new computation but returns to its previous location before its initial neighbors notice its departure, these latter nodes need to notified that  $x$  is no longer part of their computation so as not to wait indefinitely for a reply from the node.
- **hbeat**: a message of this type is emitted by a leader node. It is included in the specification simply for the analysis purposes.

In its initial state the algorithm is modeled by the process  $S$  consisting of a set of nodes who possess a leader (although this leader may be outside of their range). We write  $T$  for  $\{\mathbf{election}, \mathbf{ack0}, \mathbf{ack1}, \mathbf{leader}, \mathbf{reply}\}$ .

$$S \stackrel{\text{def}}{=} \left( \prod_{k \in K} \text{Elected} \langle b_k, \text{lead}_k \rangle \sigma_k \right) \setminus T$$

A node in  $\text{Elected}$  possesses a leader  $\text{lead}_k$  and a status  $b_k$  which records whether or not the process needs to broadcast its leader. Note that a leader node (i.e. a node with  $id = \text{lead}_k$ ) regularly sends a heartbeat to notify its neighbors of its availability. We may see in Figure 1 that if a process  $\text{Elected}$  receives an election message from one of its neighbors or, if it observes the absence of its leader (see condition  $\text{lead}_i \notin H_i$ ), it enters  $\text{InComp}$  mode, wherein it begins its quest for a new leader.

The  $\text{InComp}$  process has a number of parameters: the first records the node’s father to which eventually an **ack1** message needs to be returned (unless the node itself is the root of the tree).  $\text{src}_i$  and  $\text{lead}_i$  are the computation’s source and previous leader, whereas  $\text{max}_i$  is the maximum identifier observed by the node, initially set as the node’s own identifier. Sets  $R_i$  and  $A_i$  record the neighbors of the node that are expected to reply and the neighbors to which an **ack0** should be returned, respectively. Note that these

---


$$\begin{aligned}
& \text{Elected}\langle 0, \text{lead}_i \rangle \stackrel{\text{def}}{=} \\
& \quad \bar{b}(-, \mathbf{leader}, \text{lead}_i). \text{Elected}\langle 1, \text{lead}_i \rangle \\
& \quad + r(\mathbf{leader}, \text{lead}_j). \text{cond} ( \text{lead}_i < \text{lead}_j \triangleright \text{Elected}\langle 0, \text{lead}_j \rangle \\
& \quad \quad \quad \text{true} \triangleright \text{Elected}\langle 0, \text{lead}_i \rangle ) \\
& \quad + \bar{b}(-, \mathbf{reply}, \text{id}, \text{lead}_i). \text{Elected}\langle 0, \text{lead}_i \rangle \\
\\
& \text{Elected}\langle 1, \text{lead}_i \rangle \stackrel{\text{def}}{=} \\
& \quad r(\mathbf{leader}, \text{lead}_j). \text{cond} ( \text{lead}_i < \text{lead}_j \triangleright \text{Elected}\langle 0, \text{lead}_j \rangle \\
& \quad \quad \quad \text{lead}_i > \text{lead}_j \triangleright \text{Elected}\langle 1, \text{lead}_i \rangle \\
& \quad \quad \quad \text{true} \triangleright \text{Elected}\langle i, 0, \text{lead}_i \rangle ) \\
& \quad + \bar{b}(-, \mathbf{reply}, \text{id}, \text{lead}_i). \text{Elected}\langle 0, \text{lead}_i \rangle \\
& \quad + \text{cond} ( \text{lead}_i = \text{id} \triangleright \bar{h}(-, \mathbf{hbeat}, \text{lead}_i). \text{Elected}\langle 1, \text{lead}_i \rangle \\
& \quad \quad \quad \text{lead}_i \notin H_i \triangleright \text{InComp}\langle i, 0, i, N_i, \emptyset, i, \text{lead}_i \rangle ) \\
& \quad + r(\mathbf{election}, j, l, s). \text{cond} ( l = \text{lead}_i \triangleright \text{InComp}\langle j, 0, s, N_i - \{j\}, \emptyset, i, \text{lead}_i \rangle \\
& \quad \quad \quad \text{true} \triangleright \text{Elected}\langle i, \text{lead}_i \rangle ) \\
\\
& \text{InComp}\langle \text{NULL}, c, \text{src}_i, R_i, A_i, \text{max}_i, \text{lead}_i \rangle \stackrel{\text{def}}{=} \text{InComp}\langle i, c, \text{src}_i, R'_i, A'_i, \text{max}_i, \text{lead}_i \rangle \\
& \text{InComp}\langle \text{father}_i, c, \text{src}_i, R_i, A_i, \text{max}_i, \text{lead}_i \rangle \stackrel{\text{def}}{=} \\
& \quad \text{cond} ( c = 0 \triangleright \bar{b}(-, \mathbf{election}, \text{lead}_i, i, \text{src}_i). \text{InComp}\langle \text{father}'_i, 1, \text{src}_i, R'_i, A'_i, \text{max}_i, \text{lead}_i \rangle ) \\
& \quad + \sum_{j \in A_i} \bar{b}(j, \mathbf{ack0}, i). \text{InComp}\langle \text{father}'_i, 1, \text{src}_i, R'_i, A'_i - \{j\}, \text{max}_i, \text{lead}_i \rangle \\
& \quad + r(\mathbf{election}, j, l, s). \\
& \quad \quad \text{cond} ( l = \text{lead}_i \wedge s > \text{src}_i \triangleright \text{InComp}\langle j, 0, s, N_i - \{j\}, \emptyset, \text{max}_i, \text{lead}_i \rangle \\
& \quad \quad \quad l = \text{lead}_i \wedge s = \text{src}_i \triangleright \text{InComp}\langle \text{father}'_i, c, \text{src}_i, R'_i, A'_i \cup \{j\}, \text{max}_i, \text{lead}_i \rangle \\
& \quad \quad \quad \text{true} \triangleright \text{InComp}\langle \text{father}_i, c, \text{src}_i, R'_i, A'_i, \text{max}_i, \text{lead}_i \rangle ) \\
& \quad + r(\mathbf{ack0}, j). \text{InComp}\langle \text{father}'_i, 1, \text{src}_i, R'_i - \{j\}, A'_i, \text{max}_i, \text{lead}_i \rangle \\
& \quad + r(\mathbf{ack1}, j, s, \text{max}_j). \\
& \quad \quad \text{cond} ( s = \text{src}_i \wedge \text{max}_j > \text{max}_i \triangleright \text{InComp}\langle \text{father}'_i, c, \text{src}_i, R'_i - \{j\}, A'_i, \text{max}_j, \text{lead}_j \rangle \\
& \quad \quad \quad s = \text{src}_i \wedge \text{max}_j \leq \text{max}_i \triangleright \text{InComp}\langle \text{father}'_i, c, \text{src}_i, R'_i - \{j\}, A'_i, \text{max}_i, \text{lead}_i \rangle \\
& \quad \quad \quad \text{true} \triangleright \text{InComp}\langle \text{father}'_i, c, \text{src}_i, R'_i, A'_i, \text{max}_i, \text{lead}_i \rangle ) \\
& \quad + r(\mathbf{leader}, j, \text{lead}_j). \text{InComp}\langle \text{father}'_i, c, \text{src}_i, R'_i, A'_i, \text{max}_i, \text{lead}_i \rangle \\
& \quad + r(\mathbf{reply}, j, \text{lead}_j). \\
& \quad \quad \text{cond} ( \text{lead}_j \neq \text{lead}_i \triangleright \text{InComp}\langle \text{father}'_i, c, \text{src}_i, R'_i - \{j\}, A'_i - \{j\}, \text{max}_i, \text{lead}_i \rangle \\
& \quad \quad \quad \text{true} \triangleright \text{InComp}\langle \text{father}'_i, c, \text{src}_i, R'_i, A'_i, \text{max}_i, \text{lead}_i \rangle ) \\
& \quad + \bar{b}(-, \mathbf{reply}, i, \text{lead}_i). \text{InComp}\langle \text{father}'_i, c, \text{src}_i, R'_i, A'_i, \text{max}_i, \text{lead}_i \rangle \\
\\
& \text{Leader}\langle \text{father}_i, \text{src}_i, \text{max}_i, \text{lead}_i \rangle \stackrel{\text{def}}{=} \\
& \quad r(\mathbf{election}, j, l, s). \\
& \quad \quad \text{cond} ( l = \text{lead}_i \wedge s > \text{src}_i \triangleright \text{InComp}\langle j, 0, s, N_i - \{j\}, \emptyset, \text{max}_i, \text{lead}_i \rangle \\
& \quad \quad \quad \text{true} \triangleright \text{Leader}\langle \text{father}'_i, \text{src}_i, \text{max}_i, \text{lead}_i \rangle ) \\
& \quad + r(\mathbf{leader}, \text{lead}_j). \text{Elected}\langle 0, \text{lead}_j \rangle \\
& \quad + \bar{b}(-, \mathbf{reply}, i, \text{lead}_i). \text{Leader}\langle \text{father}'_i, \text{src}_i, \text{max}_i, \text{lead}_i \rangle \\
\\
& \text{Leader}\langle \text{NULL}, \text{src}_i, \text{max}_i, \text{lead}_i \rangle \stackrel{\text{def}}{=} \text{Elected}\langle 0, \text{max}_i \rangle
\end{aligned}$$


---

**Fig. 1.** The description of a node

sets are regularly updated according to the node's interface: we write  $R'_i = R_i \cap N_i$ ,  $A_i = A_i \cap N_i$  and  $father' = father$ , if  $father \in N_i$ , and  $NULL$ , otherwise. Finally,  $c$  is a bit that captures whether a broadcast for a leader election request has been emitted or not by the node.

Finally, node  $Leader\langle father_i, src_i, max_i, lead_i \rangle$  awaits to be notified of the component's leader by its father  $father_i$ . It recalls its computation characterized by its source and previous leader ( $src_i$  and  $lead_i$ ) as well as the maximum node it has observed from its downstream nodes,  $max_i$ . In case of the loss of its father, the node elects this maximum node as the leader of the component.

### 3.3 Verification of the Protocol

The aim of our analysis is to establish that after a finite number of topology changes, every connected component/neighborhood of the network will elect the node with the maximum identifier as its leader. Specifically, we consider an arbitrary derivative of  $S$ , namely  $S_1$ , where we assume that the topology known by all nodes of  $S_1$  is consistent with the network's state. We write  $\lceil S_1 \rceil$  for  $S_1$  with the parts of its transition system involving node movement pruned away, and we show:

**Theorem 2.**  $\lceil S_1 \rceil \approx \lceil (\prod_{k \in K} \text{Elected}\langle 1, max_k \rangle \sigma_k) \setminus T \rceil$  where  $max_k$  is the maximum node identifier in the neighborhood of node  $k$ .

In other words, our correctness criterion states that, eventually, assuming nodes stop moving, all nodes will learn a leader which is the node with the maximum identifier within their vicinity. We proceed to sketch the proof of this result which can be found in its entirety in [4].

*Sketch of Proof of Theorem 2*

Let us consider an arbitrary derivative of  $S$ . This has the form:

$$S_1 = \left( \prod_{i \in E} \text{Elected}\langle b_i, lead_i \rangle \sigma_i \mid \prod_{i \in L} \text{Leader}\langle f_i, src_i, max_i, lead_i \rangle \sigma_i \right. \\ \left. \mid \prod_{i \in C} \text{InComp}\langle f_i, c_i, src_i, R_i, A_i, max_i, lead_i \rangle \sigma_i \right) \setminus T$$

Recall that  $K$  is the set of network nodes,  $K = E \cup L \cup C$ . We partition  $K$  into the sets  $N_g$ ,  $g \in G$ , where for all  $g \in G$  and  $i, j \in N_g$ , there exists a path between  $i$  and  $j$ , whereas, if  $i \in N_g$  and  $j \notin N_g$ , there exists no path between  $i$  and  $j$ . These sets form the neighborhoods of our algorithm, where independent computations are taking place. We write  $N_g = E_g \cup L_g \cup C_g$  where  $E_g = N_g \cap E$ ,  $L_g = N_g \cap L$  and  $C_g = N_g \cap C$  and we rewrite process  $S_1$  by taking into account the network's neighborhoods as  $S_1 = (\prod_{g \in G} C_g) \setminus T$ , where:

$$C_g = \left( \prod_{i \in E_g} \text{Elected}\langle b_i, lead_i \rangle \sigma_i \mid \prod_{i \in L_g} \text{Leader}\langle f_i, src_i, max_i, lead_i \rangle \sigma_i \right. \\ \left. \mid \prod_{i \in C_g} \text{InComp}\langle f_i, c_i, src_i, R_i, A_i, max_i, lead_i \rangle \sigma_i \right) \setminus T$$

We prove the following result:

**Lemma 1.**  $\lceil C_g \rceil \approx \lceil Spec_1 \rceil$ , where  $Spec_1 \stackrel{\text{def}}{=} (\prod_{i \in N_g} \text{Elected}\langle 1, max_g \rangle \sigma_i) \setminus T$ , where  $max_g = \max\{max_i | i \in N_g\}$ .

PROOF: The proof of this result is carried out in a similar way to the proof of the static case of the algorithm [1] but lifted from value-passing CCS to the new calculus and taking into account mobility considerations. The proof deals with two important points: The first, concerns the fact that various computations can independently take place within a neighborhood of the network. This is because, according to the algorithm, computations with a distinct *lead* parameter do not merge their computations until after they elect their leader. Therefore, computation takes place on a forrest as opposed to a tree, which is the case in the static case. The second point is that the nodes comprising  $C_g$  are not themselves confluent, therefore we cannot conclude the confluence of  $C_g$  by construction. To deal with this, as in [1], the proof takes place in two steps. In the first step, we consider a simplification of  $C_g$ ,  $D_g$ , where each node  $x$  in  $D_g$  is associated with a specific father-node being the unique node that  $x$  can accept as its father. We show that, by construction and Lemma 1,  $D_g$  is a confluent process and we exhibit an execution which leads to  $Spec_1$ . By  $D_g$ 's confluence and the fact that  $D_g \Longrightarrow Spec_1$ , we conclude that  $D_g \approx Spec_1$ . For the second step of the proof we show that whenever  $C_g \Longrightarrow F$  where exists a  $D_g$  (i.e. an assignment of fathers to nodes) that is similar to  $F$ . This allows us to deduce that in fact  $C_g \approx Spec_1$  as required.  $\square$

Bearing in mind that network  $S_1$  is a composition of the components  $C_g$ , each concerning a distinct communication neighborhood of the network, confluence arguments allow us to deduce the following:

**Lemma 2.**  $\lceil S_1 \rceil \approx \lceil Spec_2 \rceil$ , where  $Spec_2 \stackrel{\text{def}}{=} (\prod_{g \in G} \prod_{i \in N_g} \text{Elected}\langle 1, max_i \rangle \sigma_i) \setminus T$  where  $max_g = \max\{max_i | i \in N_g\}$ .

Now two cases exist: Either the  $max_g$  are nodes located in the neighborhoods  $C_g$ , or they are nodes that, although existed in the components in the past, they no longer reside in the component. By the same arguments applied for  $S_1$ , and the fact that all nodes of the network will now begin computation with  $max = id$ , we conclude that:

**Lemma 3.**  $\lceil S_2 \rceil \approx \lceil Spec_3 \rceil$ , where  $Spec_3 \stackrel{\text{def}}{=} (\prod_{g \in G} \prod_{i \in N_g} \text{Elected}\langle 1, max_g \rangle \sigma_i) \setminus T$  where  $max_g = \max\{id_i | i \in N_g\}$ .

This gives us the correctness of Theorem 2 and completes the proof.  $\square$

## 4 Conclusions

In this paper we have introduced a process calculus for reasoning about MANET protocols. The salient aspect of the calculus is its ability to encode protocols that manage topology (e.g. discovery of neighbors) thus enabling one to focus on the details of an algorithm at hand while abstracting from topology computations. For our case study, this aspect was also useful for describing the heartbeat protocol employed by the network's nodes to re-iterate the existence of a leader. We have also developed a theory of confluence for our process calculus. This theory is exceptionally simple when compared

with similar theories for e.g. the  $\pi$ -calculus: the absence of channels for communication and the broadcasting style of message transmission has removed a number of considerations relating to confluence preservation resulting in simple and easy to employ results. We have illustrated the applicability of the new formalism via the analysis of a leader-election MANET protocol, which was in fact the inspiration for a great part of the development: the static version of the algorithm was proved correct using a value-passing CCS in [1]. However, modeling the mobile version of the algorithm in the same style required of us to draw a number of unnatural assumptions with regards to the topology changes of the network, while proving correctness was considerably more demanding than the present approach, even under the made assumptions. In current work we are further investigating the usefulness of our results in other topology-dependent algorithms, including a MANET routing algorithm.

## References

1. Ch. Georgiou, M. Gelastou, and A. Philippou. On the application of formal methods for specifying and verifying distributed protocols. In *Proceedings of NCA'06*, pages 195–204. IEEE Computer Society, 2008.
2. J. Ch. Godskesen. A calculus for mobile ad-hoc networks with static location binding. *Electronic Notes of Theoretical Computer Science*, 242(1):161–183, 2009.
3. J. F. Groote and M. P. A. Sellink. Confluence for process verification. In *Proceedings of CONCUR'95*, LNCS 962, pages 152–168, 2005.
4. D. Kouzapas and A. Philippou. A process calculus for systems with dynamic topology. Technical Report TR-10-01, University of Cyprus, 2010. Also available as <http://www.cs.ucy.ac.cy/annap/tr-10-01.pdf>.
5. X. Liu and D. Walker. Confluence of processes and systems of objects. In *Proceedings of TAPSOFT'95*, LNCS 915, pages 217–231, 1995.
6. M. Merro. An observational theory for mobile ad hoc networks. *Information and Computation*, 207(2):194–208, 2009.
7. N. Mezzetti and D. Sangiorgi. Towards a calculus for wireless systems. *Electronic Notes of Theoretical Computer Science*, 158:331–353, 2006.
8. R. Milner. *A Calculus of Communicating Systems*. Springer, 1980.
9. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts 1 and 2. *Information and Computation*, 100:1–77, 1992.
10. U. Nestmann. *On Determinacy and Non-determinacy in Concurrent Programming*. PhD thesis, University of Erlangen, 1996.
11. A. Philippou and G. Michael. Verification techniques for distributed algorithms. In *Proceedings of OPODIS'06*, LNCS 4305, pages 172–186, 2006.
12. A. Philippou and D. Walker. On transformations of concurrent object programs. In *Proceedings of CONCUR'96*, LNCS 1119, pages 131–146, 1996.
13. A. Philippou and D. Walker. On confluence in the  $\pi$ -calculus. In *Proceedings of ICALP'97*, LNCS 1256, pages 314–324, 1997.
14. A. Singh, C. R. Ramakrishnan, and S. A. Smolka. A process calculus for mobile ad hoc networks. In *Proceedings of COORDINATION'08*, LNCS 5052, pages 296–314, 2008.
15. C. Tofts. *Proof Methods and Pragmatics for Parallel Programming*. PhD thesis, University of Edinburgh, 1990.
16. S. Vasudevan, J. Kurose, and D. Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. In *Proceedings of ICNP'04*, pages 350–360. IEEE Computer Society, 2004.

## A Appendix

**Table 3. Structural congruence relation**

---

(N1) $M \equiv M 0$	(N2) $M_1 M_2 \equiv M_2 M_1$
(N3) $(M_1 M_2) M_3 \equiv M_1 (M_2 M_3)$	(N4) $M \setminus T - \{t\} \equiv M \setminus T$ if $t \notin \text{types}(M)$
(N5) $M \setminus T \equiv (M \setminus T - \{t\}) \setminus \{t\}$	(N6) $M_1 \equiv M_2$ if $M_1 \equiv_\alpha M_2$
(N7) $M_1 \setminus \{t\}   M_2 \equiv (M_1 \setminus M_2) \setminus \{t\}$ if $t \notin \text{types}(M_2)$	
(P1) $P_1 + P_2 \equiv P_2 + P_1$	(P2) $(P_1 + P_2) + P_3 \equiv P_1 + (P_2 + P_3)$

---