A Process Calculus for Dynamic Networks *

Dimitrios Kouzapas[1] and Anna Philippou[2]

[1] Imperial College, London, UK. dk208@doc.ic.ac.uk
[2] University of Cyprus, Cyprus. annap@cs.ucy.ac.cy

Abstract

In this paper we propose a process calculus framework for dynamic networks in which the network topology may change as computation proceeds. The proposed calculus allows one to abstract away from neighborhood-discovery computations and it contains features for broadcasting at multiple transmission ranges and 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 verification of a leader-election algorithm for mobile ad hoc networks.

1 Introduction

Distributed and wireless systems present today one of the most challenging areas of research in computer science. Their high complexity, 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 the π-calculus [10], 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, the Calculus for Systems with Dynamic Topology, CSDT, is inspired by works which have previously appeared in the literature [2, 5, 7, 16, 18] on issues such as broadcasting, movement and separating between a node's control and topology information. However, our calculus extends these works by considering two additional MANET features. The first concerns the ability of MANETs to broadcast messages at different transmission levels. This is a standard feature of mobile ad hoc networks; a node may choose to broadcast a message at a high transmission power in order to communicate with a wider set of nodes, as required by a protocol or application, or it may choose to use a low transmission power in order to conserve the node's power. The second feature concerns that of neighbor discovery. As computation proceeds and nodes move in and out of each other's transmission range, each node attempts to remain aware of its connection topology in order to successfully complete its tasks (e.g. routing). To achieve this, neighbor discovery protocols are implemented and

*Technical Report TR-10-01, Department of Computer Science, University of Cyprus. An extended abstract of the article is currently under submission.

run which, which typically involve periodically emitting "hello" messages and acknowledging such messages received by one's neighbors. Although it is possible that at various points in time a node does not have the precise information regarding its neighbors, these protocols aim to ensure that the updated topology is discovered and that the node adjusts its behavior accordingly so that correct behavior is achieved. Thus, when one is called to model and verify a MANET protocol, it is important to take into account that neighbor-discovery is running in the background and that the neighbor-information available to a node may not be correct at all times. To handle this 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 of 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. for reasoning about timed behaviors).

CSDT allows for reasoning about both of these aspects of behavior. To begin with, it allows nodes to broadcast information at two different transmission levels. In particular, the transmission level of a broadcast is encoded as a parameter of broadcasted messages. We point out that this could easily be extended to a wider range of transmission levels, as discussed further in Section 4. Here we opt to implement two levels for the sake of simplicity and because this is already sufficient for the case study we consider. Regarding the issue of neighborhood discovery, the semantics of CSDT contain rules that mimic the behavior of a neighborhood-discovery protocol. To achieve this CSDT equips nodes with knowledge of their believed (and not necessarily actual) sets of neighbors. These sets capture the belief of a node regarding its neighbors which is not necessarily correct and which is continuously updated, as needed. This continuous observation of the changes in a node's topology is intended to capture the way a neighborhood-discovery algorithm operates and gradually discerns changes in the connectivity of a node. Furthermore, these neighbors sets are accessible from the control part of a node and may affect the flow of the node's execution. We believe that this feature is both realistic as well as convenient for modeling and analyzing ad hoc network protocols. A final novelty of CSDT which has been introduced for facilitating verification within the calculus, is the introduction of a hiding construct 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 [19, 3, 4, 14, 12, 15, 13]. 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.

Related Work. Several process calculi have recently been proposed for dynamic networks. In [5] the process calculus CMN is introduced as a basis for studying the observational theory of mobile ad hoc networks. Like CSDT, a node in CMN is a named, located process which communicates via broadcasting within its transmission range. A difference in the two approaches concerns the treatment of the notion of a location: while in CMN locations can be viewed as values in a coordinate systems and neighborhood is computed via distance function, in CSDT, locations and their interconnections are given in the form of a graph. Unlike CSDT, CMN features both stationary and mobile nodes and it caters for message loss. On the other hand, CSDT allows to describe unicast communication. In [11], the authors propose CBS# where the recipients of a transmission are determined using a graph representation of node localities. They use their framework as a basis for performing security analysis of MANET communication protocols. The ω-calculus, an extension of the π-calculus is the object of study in [18]. A key feature of the ω-calculus is the separation of a node's control component from the description of its transmission range. This latter is implemented by annotating a process with a set of groups which constitute the connected components of the network to which the node belongs. The ω-calculus enables unicast communication and caters for message loss. In CMAN [2], each node is associated with its location and the set of locations to which it may communicate over uni-directional links. This explicit handling of the connection topology via individual nodes poses some difficulties in defining non-contextual equivalences relations, though the obtained results appear to be adequate for the case studies considered. Finally, [7] defines a lower-level calculus to describe interference in wireless systems and [6] studies a timed process calculus for wireless networks exposed to collisions.

As it has already been discussed, CSDT extends all of these frameworks by allowing to broadcast at different transmission levels, by associating nodes with their believed, as opposed to actual sets of neighbors which are updated by semantic rules that mimic the behavior of a neighbor discovery protocol and via a restriction construct.

Contribution. The contribution of our work is summarized as follows:

1. We define a new process calculus for reasoning about dynamic networks. This calculus introduces a number of new ideas which have been selected in view of facilitating the modeling and the analysis of mobile ad hoc network protocols.

2. We develop the theory of confluence in the context of our calculus. This is the first such theory for process calculi featuring broadcast communication.

3. We provide a correctness proof of a non-trivial leader-election protocol proposed in [20] for mobile ad hoc networks.

The remainder of the paper is structured as follows. In Section 2 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.

2 The Process Calculus

In our Calculus for Systems with Dynamic Topology, CSDT, we consider a system as a set of nodes operating in space. Each node possesses a physical location and a unique identifier. Movement is modeled as the change in the location of a node, with the restriction that

the originating and the destination locations are neighboring locations. As an example, Figure 1(a) portrays a network comprising of four nodes with identifiers 1-4 where node i resides at location ℓ_i, whereas Figure 1(b) portrays the same network where node 4 has moved from location ℓ_4 to location ℓ_5.

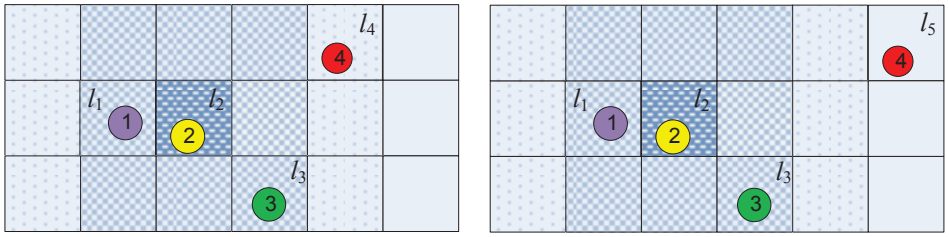

Figure 1: Wireless network before (a) and after (b) the move of node 4

Nodes in CSDT can communicate with each other by broadcasting messages. Broadcasting may take place at different transmission levels, thus allowing a node to adjust the range of a transmission, as required by an underlying protocol and/or for power-saving purposes. Specifically, in CSDT we consider two transmission levels, a normal level and a high level. Messages sent at the high transmission level are transmitted to a wider set of locations in comparison to those sent at the normal transmission level.

Neighbor discovery, that is, determining which nodes fall within the transmission range of a node, is a building block of network protocols and applications. To facilitate the reasoning about such protocols we embed in the semantics of our calculus rules that mimic the behavior of a neighborhood discovery algorithm, which observes changes in the network's topology and, specifically, the departure and arrival of nodes within the normal and high transmission ranges of a node. To capture this information, each node is associated with two sets of nodes, N and H, which are the sets of nodes believed to be within the normal and high transmission ranges of a node, respectively. Specifically, we write $P{:}[\![id, \ell, N, H]\!]$, for describing a node running code P with unique identifer id, located at physical location ℓ, believed normal-range neighbors N, and believed high-range neighbors H.

Returning to the example of Figure 1, let us assume that messages broadcasted at the normal transmission level at some location ℓ are received by all cells/locations at distance 1 from ℓ, and messages broadcasted at the high transmission level are received by all nodes within distance 2 from ℓ. Then, for example, node 2 in Figure 1(a) has as its normal-range neighbors nodes $\{1, 3\}$ and as its high-range neighbors nodes $\{1, 2, 4\}$. This network can be modeled by the following process, where process P_i represents the code running at node i:

$$P_1{:}[\![1, \ell_1, \{2\}, \{2, 3\}]\!] \mid P_2{:}[\![2, \ell_2, \{1, 3\}, \{1, 3, 4\}]\!] \mid P_3{:}[\![3, \ell_3, \{2\}, \{1, 2, 4\}]\!] \mid P_4{:}[\![4, \ell_4, \emptyset, \{2, 3\}]\!]$$

The network in Figure 1(b) where node N_4 has moved from location 4 to location 5, and assuming that the neighbor-discovery protocol has not yet had a chance to execute, corresponds to the process:

$$P_1{:}[\![1, \ell_1, \{2\}, \{2, 3\}]\!] \mid P_2{:}[\![2, \ell_2, \{1, 3\}, \{1, 3, 4\}]\!] \mid P_3{:}[\![3, \ell_3, \{2\}, \{1, 2, 4\}]\!] \mid P_4{:}[\![4, \ell_5, \emptyset, \{2, 3\}]\!]$$

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 CSDT. We assume a set of node identifiers $\mathcal{I}$ ranged over

by id, i, j, and a set of physical locations $\mathcal{L}$ ranged over by ℓ, ℓ' and we say that two locations ℓ, ℓ' are neighboring locations if $(\ell, \ell') \in Nb$, where $Nb \subseteq \mathcal{L} \times \mathcal{L}$. Furthermore, we assume a set of transmission levels $\{\mathrm{n}, \mathrm{h}\}$ and associated with these levels the functions $\mathrm{range}_{\mathrm{n}} : \mathcal{L} \to 2^{\mathcal{L}}$ and $\mathrm{range}_{\mathrm{h}} : \mathcal{L} \to 2^{\mathcal{L}}$ which, given a location, return the locations that fall within its normal and high transmission levels, respectively. Note that these functions, need not take a symmetric view on locations and may be defined so as to yield unidirectional links. Furthermore, we assume a set of special labels $\mathcal{T}$. Elements of $\mathcal{T}$ are mere keywords appended to messages indicating the message type.

In addition, 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 $f(e_1, \ldots e_n)$ where f is a function from a set of functions (e.g. logical connectives, set operators and arithmetic operators), and the e_i are terms. We say that a term is *closed* if it contains no variables. The evaluation relation $\twoheadrightarrow$ for closed terms is defined in the expected manner. We write $\tilde{r}$ for a tuple of syntactic entities $r_1, \ldots, r_n$.

Finally, we assume a set of process constants $\mathcal{C}$, denoted by C, each with an associated definition of the form $C\langle\tilde{x}\rangle \stackrel{\text{def}}{=} P$, where P may contain occurrences of C, as well as other constants. Based on this basic entities, the syntax of of CSDT is given in Table 1, where $T \subseteq \mathcal{T}$.

Table 1: **The Syntax**

Actions:	η	::=	$\overline{b}(w, t, \tilde{v}, tl)$	broadcast
		|	$r(t, \tilde{x})$	input
Processes:	P	::=	$\mathbf{0}$	Inactive process
		|	$\eta.P$	Action prefix
		|	$P_1 + P_2$	Nondeterministic Choice
		|	$\mathsf{cond}\ (e_1 \rhd P_1, \ldots, e_n \rhd P_n)$	Conditional
		|	$C\langle\tilde{v}\rangle$	Process Constant
Networks:	M	::=	$\mathbf{0}$	
		|	$P{:}\sigma$	Located Node
		|	$M_1 \mid M_2$	Parallel Composition
		|	$M \backslash T$	Restriction
Interfaces:	σ	::=	$[\![id, \ell, N, H]\!]$	

There are two types of actions in CSDT. A broadcast action $\overline{b}(w, t, \tilde{v}, tl)$ is a transmission at transition level tl, of type $t \in \mathcal{T}$ of the tuple $\tilde{v}$ with intended recipients w, 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. An input action $r(t, \tilde{x})$ represents a receipt of a message $\tilde{x}$ of type t. In turn, a process can then be inactive, an action-prefixed process, the nondeterministic choice between two processes, a process constant or a conditional process. The conditional process $\mathsf{cond}\ (e_1 \rhd P_1, \ldots, e_n \rhd P_n)$ 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 σ is the node's topology information which we call its *interface*. An interface σ contains the node identifier id, its location ℓ as well as its normal and heartbeat neighbors N and H, according to its current knowledge. We allow the control part of a located process $P{:}\sigma$,

namely P, to access information mentioned in the interface σ, by using the special labels id, l, N and H, thus allowing the control part of a process to express dependencies on the node's neighborhood information. For example, an expression "$4 \in$ N" 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 $M_1 \mid M_2$, or a restricted network $M\backslash T$. In $M\backslash T$ the scope of messages of all types in T is restricted to network M: components of M may exchange messages of these types to interact with one another but not with M's environment. Note that a type t restricted within a node P is intended to be distinct from any other occurrences of t in another network M within some wider network $P\backslash\{t\} \mid M$. To avoid name collisions on types, we consider α-conversion on processes as the renaming of types that are bound by some restriction and we say that P and Q are α-equivalent, $P \equiv_\alpha Q$, if P and Q differ by a renaming of bound types.

In what follows, given an interface σ, we write $\mathrm{l}(\sigma)$ and $\mathrm{id}(\sigma)$ for the location and the identifier mentioned in σ, respectively. Moreover, we use $\mathrm{types}(M)$ to denote the set of all types occurring in activities of M.

2.2 The Semantics

The semantics of CSDT is defined in terms of a structural congruence relation $\equiv$, which can be found in Table 2, and a structural operational semantics, which is given in Tables 3 and 4.

Table 2: **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)	$M\backslash T - \{t\} \equiv M\backslash T$ if $t \notin \mathrm{types}(M)$
(N5)	$M\backslash T \equiv (M\backslash T - \{t\})\backslash\{t\}$	(N6)	$M_1 \equiv M_2$ if $M_1 \equiv_\alpha M_2$
(N7)	$M_1\backslash\{t\}\|M_2 \equiv (M_1\|M_2)\backslash\{t\}$ if $t \notin \mathrm{types}(M_2)$		

The rules of Table 3 describe the behavior of located processes in isolation whereas the rules in Table 4 the behavior of networks. A transition of P (or M) has the form $P \xrightarrow{\alpha} P'$, specifying that P can perform action α and evolve into P' where α can have one of the following forms:

- $\overline{b}(w, t, \tilde{v}, tl, \ell)$ denotes a broadcast to recipients w of a message $\tilde{v}$ of type t at transmission level tl, taking place at location ℓ.
- $r(id, t, \tilde{v}, \ell)$ denotes a receipt by node id of a message $\tilde{v}$ of type t, taking place at location ℓ.
- $r?(id, t, \tilde{v}, \ell)$ denotes an advertisement by node id that it is willing to receive a message $\tilde{v}$ of type t, at location ℓ.
- τ and μ denote two types of unobservable actions in the calculus. Action τ is associated with the effect of restriction (rule (Hide2), Table 4) and μ is associated with

the movement of nodes (rule (Move), Table 3) as well as with updates of neighborhood information (rules (InScopeN), (InScopeH), (OutofScopeN) and (OutofScopeH), Table 4).

We let Act denote the set of all actions and let α and β range over Act and we write type(α) for the type of an $\alpha \in Act - \{\tau, \mu\}$.

Table 3: **Transition rules for located nodes**

(BCast)	$(\overline{b}(w,t,\tilde{v},tl).P){:}\sigma \overset{\overline{b}(w,t,\tilde{v},tl,\mathrm{l}(\sigma))}{\longrightarrow} P{:}\sigma$	(Rec)	$(r(t,\tilde{x}).P){:}\sigma \overset{r(\mathrm{id}(\sigma),t,\tilde{v},\mathrm{l}(\sigma))}{\longrightarrow} P\{\tilde{v}/\tilde{x}\}{:}\sigma$
(Const)	$\dfrac{[P\{\tilde{v}/\tilde{x}\}]{:}\sigma \xrightarrow{\alpha} P'{:}\sigma}{[C\langle\tilde{v}\rangle]{:}\sigma \xrightarrow{\alpha} P'{:}\sigma} \quad C\langle\tilde{x}\rangle \overset{\mathrm{def}}{=} P$	(Sum)	$\dfrac{P_i{:}\sigma \xrightarrow{\alpha} P_i'{:}\sigma}{(P_1+P_2){:}\sigma \xrightarrow{\alpha} P_i'{:}\sigma}, \ i \in \{1,2\}$
(Cond)	$\dfrac{P_i{:}\sigma \xrightarrow{\alpha} P_i'{:}\sigma}{(\mathsf{cond}\ (e_1 \rhd P_1, \ldots, e_n \rhd P_n)){:}\sigma \xrightarrow{\alpha} P_i'{:}\sigma} \quad e_i \twoheadrightarrow \mathsf{true}, \quad \forall j < i, e_j \twoheadrightarrow \mathsf{false}$		
(Move)	$\dfrac{(\ell,\ell') \in Nb}{P{:}[\![i,\ell,N,H]\!] \xrightarrow{\mu} P{:}[\![i,\ell',N,H]\!]}$		

It is worth noting that in the first two rules of Table 3, communication actions are extended with 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 4.

Moving our attention to the rules of Table 4, we point out that the first four rules implement the underlying neighborhood 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 which have exited these transmission ranges are removed from the appropriate sets in its interface. In all four cases a μ internal action takes place signifying a move of the neighborhood-discovery protocol.

The two (Broadcast) rules which follow give semantics to broadcasting within the language. They employ the compatibility function comp, where comp(w,i) is evaluated to true only identifier i is compatible with intended recipient w: comp$(w,i) = (w =' -' \vee w = i)$. Axiom (Broadcast1) specifies that, if a broadcast is available and there exists is a compatible recipient within the range of the broadcast, the message is received and the broadcast message is propagated. On the other hand, if a broadcast is available but there is no appropriate receiver then, again, the broadcast is propagated in the network (Broadcast2). Note that rule (Broadcast2) (as well as (Receive)) is defined in terms of the inability of executing an action via the use of relation "discards", $\overset{\alpha}{\not\rightarrow}$, which is defined inductively in Table 5.

Moving on to rule (Receive), we point out that a network must advertise the fact that it may receive an input. This is necessary, otherwise an inactive network and a network such as $M \overset{\mathrm{def}}{=} [r(t,\tilde{x}).P]{:}\sigma_1 \mid [r(t,\tilde{y}).Q]{:}\sigma_2$ would have exactly the same transition systems when clearly they would yield distinct behaviors when placed in parallel with a network such as $[\overline{b}(-,t,\tilde{v},\mathrm{n}).S]{:}\sigma$ (affecting compositionality in the calculus). Now, if rule (Receive) was enunciated via a normal receive action instead of $r?$, we would have: $M \overset{r(\mathrm{id}(\sigma),t,\tilde{v},\mathrm{l}(\sigma))}{\longrightarrow}$ $[P\{\tilde{v}/\tilde{x}\}]{:}\sigma_1 \mid [r(t,\tilde{y}).Q]{:}\sigma_2$ and subsequently $M \mid [\overline{b}(-,t,\tilde{v},\mathrm{n}).S]{:}\sigma \overset{\overline{b}(-,t,\tilde{v},\mathrm{n},\mathrm{l}(\sigma))}{\longrightarrow} ([P\{\tilde{v}/\tilde{x}\}]{:}\sigma_1 \mid [r(t,\tilde{y}).Q]{:}\sigma_2) \mid S{:}\sigma$, which is not the intended meaning of a broadcast communication. In-

Table 4: **Transition rules for networks**

(InScopeN)	$\dfrac{\mathrm{l}(\sigma) \in \mathrm{range_n}(\ell),\ \mathrm{id}(\sigma) \notin N}{P{:}[\![id, \ell, N, H]\!] \mid Q{:}\sigma \xrightarrow{\mu} P{:}[\![id, \ell, N \cup \{\mathrm{id}(\sigma)\}, H]\!] \mid Q{:}\sigma}$
(OutofScopeN)	$\dfrac{\mathrm{l}(\sigma) \notin \mathrm{range_n}(\ell),\ \mathrm{id}(\sigma) \in N}{P{:}[\![id, \ell, N, H]\!] \mid Q{:}\sigma \xrightarrow{\mu} P{:}[\![id, \ell, N - \{\mathrm{id}(\sigma)\}, H]\!] \mid Q{:}\sigma}$
(InScopeH)	$\dfrac{\mathrm{l}(\sigma) \in \mathrm{range_h}(\ell),\ \mathrm{id}(\sigma) \notin H}{P{:}[\![id, \ell, N, H]\!] \mid Q{:}\sigma \xrightarrow{\mu} P{:}[\![id, \ell, N, H \cup \{\mathrm{id}(\sigma)\}]\!] \mid Q{:}\sigma}$
(OutofScopeH)	$\dfrac{\mathrm{l}(\sigma) \notin \mathrm{range_h}(\ell),\ \mathrm{id}(\sigma) \in H}{P{:}[\![id, \ell, N, H]\!] \mid Q{:}\sigma \xrightarrow{\mu} P{:}[\![id, \ell, N, H - \{\mathrm{id}(\sigma)\}]\!] \mid Q{:}\sigma}$
(Broadcast1)	$\dfrac{M_1 \xrightarrow{\overline{b}(w,t,\tilde{v},tl,\ell)} M_1', M_2 \xrightarrow{r(id,t,\tilde{v},\ell')} M_2', \mathrm{comp}(w, id), \ell' \in \mathrm{range}_{tl}(\ell)}{M_1 \mid M_2 \xrightarrow{\overline{b}(w,t,\tilde{v},tl,\ell)} M_1' \mid M_2'}$
(Broadcast2)	$\dfrac{M_1 \xrightarrow{\overline{b}(w,t,\tilde{v},tl,\ell)} M_1' \text{ and } M_2 \xrightarrow{r(id,t,\tilde{v},\ell')\!\!\!\!\rightarrow} \forall id', \ell' \cdot \mathrm{comp}(w, id'), \ell' \in \mathrm{range}_{tl}(\ell)}{M_1 \mid M_2 \xrightarrow{\overline{b}(w,\ell,l,t,\tilde{v})} M_1' \mid M_2}$
(Receive)	$\dfrac{M_1 \xrightarrow{r(id,\ell,t,\tilde{v})} M_1' \text{ and } M_2 \xrightarrow{\overline{b}(w,t,\tilde{v},tl,\ell')\!\!\!\!\rightarrow} \forall w, \ell', tl \cdot \mathrm{comp}(w, id), \ell \in \mathrm{range}_{tl}(\ell')}{M_1 \mid M_2 \xrightarrow{r?(id,\ell,t,\tilde{v})} M_1' \mid M_2}$
(Hide1)	$\dfrac{M \xrightarrow{\alpha} M' \text{ and } \mathrm{type}(\alpha) \notin T}{M\backslash T \xrightarrow{\alpha} M'\backslash T}$
(Hide2)	$\dfrac{M \xrightarrow{\alpha} M' \text{ and } \mathrm{type}(\alpha) \in T}{M\backslash T \xrightarrow{\tau} M'\backslash T}$
(Par)	$\dfrac{M_1 \xrightarrow{\alpha} M_1', \quad \alpha \in \{\tau, \mu\}}{M_1 \mid M_2 \xrightarrow{\alpha} M_1' \mid M_2}$
(Struct)	$\dfrac{M_1 \equiv M_2, M_2 \xrightarrow{\alpha} M_2', M_2' \equiv M_1'}{M_1 \xrightarrow{\alpha} M_1'}$

stead, only located processes can emit an $r(\ldots)$ action and for the system above we have:

$$M \mid [\overline{b}(-, t, \tilde{v}, \mathrm{n}).S]{:}\sigma \equiv [r(t, \tilde{x}).P]{:}\sigma_1 \mid ([r(t, \tilde{y}).Q]{:}\sigma_2 \mid [\overline{b}(-, t, \tilde{v}, \mathrm{n}).S]{:}\sigma)$$

and since $[r(t, \tilde{y}).Q]{:}\sigma_2 \mid [\overline{b}(-, t, \tilde{v}, \mathrm{n}).S]{:}\sigma \xrightarrow{\overline{b}(-,t,\tilde{v},\mathrm{n},\mathrm{l}(\sigma))} [Q\{\tilde{v}/\tilde{y}\}]{:}\sigma_2 \mid S{:}\sigma$, then

$$[r(t, \tilde{x}).P]{:}\sigma_2 \mid ([r(t, \tilde{y}).Q]{:}\sigma_2 \mid [\overline{b}(-, t, \tilde{v}, \mathrm{n}).S]{:}\sigma) \xrightarrow{\overline{b}(-,t,\tilde{v},\mathrm{n},\mathrm{l}(\sigma))} [P\{\tilde{v}/\tilde{x}\}]{:}\sigma_1 \mid ([Q\{\tilde{v}/\tilde{y}\}]{:}\sigma_2 \mid S{:}\sigma)$$

and by (Struct), $M \mid [\overline{b}(-, t, \tilde{v}, \mathrm{n}).S]{:}\sigma \xrightarrow{\overline{b}(-,t,\tilde{v},\mathrm{n},\mathrm{l}(\sigma))} ([P\{\tilde{v}/\tilde{x}\}]{:}\sigma_1 \mid [Q\{\tilde{v}/\tilde{y}\}]{:}\sigma_2) \mid S{:}\sigma$.

For restriction (rules (Hide1) and (Hide2)) we observe that the effect of restricting a set of types T within a process is to transform all actions involving messages of the restricted set into internal actions.

2.3 Bisimulation

In the next three sections we build some machinery for reasoning about networks described in CSDT. First, let us recall that Q is a *derivative* of P, if there exist actions $\alpha_1, \ldots, \alpha_n$,

Table 5: **The "discards" relation**

(D1)	$\dfrac{\alpha \in Act - \{\mu\}}{\mathbf{0}{:}\sigma \overset{\alpha}{\not\to}}$	(D2)	$\dfrac{\alpha \in Act - \{\tau, \mu\}}{(\tau.P){:}\sigma \overset{\alpha}{\not\to}}$
(D3)	$\dfrac{\alpha \notin \{\bar{b}(w, t, \tilde{v}, tl, \mathrm{l}(\sigma)), \mu\}}{(\bar{b}(w, t, \tilde{v}, tl).P){:}\sigma \overset{\alpha}{\not\to}}$	(D4)	$\dfrac{\alpha \notin \{r(\mathrm{id}(\sigma), t, \tilde{v}, \mathrm{l}(\sigma)), \mu\}}{(r(t, \tilde{x}).P){:}\sigma \overset{\alpha}{\not\to}}$
(D5)	$\dfrac{M_1 \overset{\alpha}{\not\to} \ \wedge \ M_2 \overset{\alpha}{\not\to}}{M_1 \mid M_2 \overset{\alpha}{\not\to}}$	(D6)	$\dfrac{P_1{:}\sigma \overset{\alpha}{\not\to} \ \wedge \ P_2{:}\sigma \overset{\alpha}{\not\to}}{(P_1 + P_2){:}\sigma \overset{\alpha}{\not\to}}$
(D7)	$\dfrac{P_i{:}\sigma \overset{\alpha}{\not\to}}{(\mathsf{cond}\ (e_1 \rhd P_1, \ldots, e_n \rhd P_n)){:}\sigma \overset{\alpha}{\not\to}}$	$e_i \twoheadrightarrow \mathsf{true}, \ \forall j < i, e_j \twoheadrightarrow \mathsf{false}$	
(D8)	$\dfrac{[P{:}\{\tilde{v}/\tilde{x}\}]{:}\sigma \overset{\alpha}{\not\to}}{[C\langle\tilde{v}\rangle]{:}\sigma \overset{\alpha}{\not\to}}, \ C\langle\tilde{x}\rangle \overset{\mathrm{def}}{=} P$	(D9)	$\dfrac{\alpha \in Act}{\mathbf{0} \overset{\alpha}{\not\to}}$
(D10)	$\dfrac{M \overset{\alpha}{\not\to} \text{ or } \mathrm{type}(\alpha) \in T}{M\backslash T \overset{\alpha}{\not\to}}$		

$n \geq 0$, such that $P \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q$. Moreover, we define weak transitions as follows: given an action α we write $P \Longrightarrow Q$ for $P(\xrightarrow{\tau,\mu})^* Q$, $P \overset{\alpha}{\Longrightarrow} Q$ for $P \Longrightarrow \xrightarrow{\alpha} \Longrightarrow Q$, and $P \overset{\hat{\alpha}}{\Longrightarrow} Q$ for $P \Longrightarrow Q$ if $\alpha \in \{\tau, \mu\}$, $P \overset{r(id,\ell,t,\tilde{v})}{\Longrightarrow} Q$, if $\alpha = r?(id, \ell, t, \tilde{v})$, and $P \overset{\alpha}{\Longrightarrow} Q$ otherwise.

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

Definition 2.1 *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 $Q \overset{\hat{\alpha}}{\Longrightarrow} Q'$ and $P' \approx Q'$.

We may prove that bisimilarity is a congruence relation:

Lemma 2.2 If $P{:}\sigma \approx Q{:}\sigma$ and $P_i{:}\sigma \approx Q_i{:}\sigma$ for $i \in \{1, \ldots, n\}$, then the following hold:

1. $(\eta.P){:}\sigma \approx (\eta.Q){:}\sigma$
2. $(P_1 + Q){:}\sigma \approx (P_2 + Q){:}\sigma$
3. $\mathsf{cond}\ (e_1 \rhd P_1, \ldots, e_n \rhd P_n){:}\sigma \approx \mathsf{cond}\ (e_1 \rhd Q_1, \ldots, e_n \rhd Q_n){:}\sigma$

Moreover, if $M_1 \approx M_2$:

4. $M_1|M \approx M_2|M$
5. $M_1\backslash T \approx M_2\backslash T$

PROOF: Properties 1-3 as well as 5 are easy to prove by establishing the appropriate relations. Below we present the proof of 4. Consider

$$\mathcal{R} = \{(M_1|M, M_2|M) | M_1 \approx M_2\}$$

Let $(M_1|M, M_2|M) \in \mathcal{R}$ and suppose $M_1|M \xrightarrow{\alpha} M_1'|M'$. We need to show that $M_2|M \overset{\hat{\alpha}}{\Longrightarrow} M_2'|M'')$, where $(M_1'|M', M_2'|M'') \in \mathcal{R}$. The proof is a case analysis on the possible transitions.

- Suppose $\alpha \in \{\tau, \mu\}$ and $M_1|M \xrightarrow{\alpha} M_1'|M$ where $M_1 \xrightarrow{\alpha} M_1'$. Then, $M_2 \Longrightarrow M_2' \approx M_1'$, $M_2|M \Longrightarrow M_2'|M$ where $(M_1'|M, M_2'|M) \in \mathcal{R}$ as required. The proof is similar if $M \xrightarrow{\tau} M'$.

- Suppose $\alpha = r?(id, t, \tilde{v}, \ell)$ where $M_1 \xrightarrow{\alpha} M_1'$ and $M \not\xrightarrow{\beta}$ for any broadcasting action β which matches α. Since $M_1 \approx M_2$, $M_2 \stackrel{\alpha}{\Longrightarrow} M_2' \approx M_1'$ and obviously $(M_1'|M, M_2'|M) \in \mathcal{R}$. The case is similar in the case that action α is executed by M.

- Suppose $\alpha = \overline{b}(w, t, \tilde{v}, tl, \ell)$, $M_1|M \xrightarrow{\alpha} M_1'|M'$, where $M_1 \xrightarrow{\alpha} M_1'$ and

$$M \equiv (M_{1,1}|(M_{1,2}|\ldots|(M_{i,k}|M^*)\ldots))\backslash T,$$

where $M_{1,i} \stackrel{r(i,t,\tilde{v},\ell_i)}{\longrightarrow} M_{1,i}'$, $\mathrm{comp}(w, i)$, $\ell_i \in \mathrm{range}_{tl}(\ell)$, $M^* \not\xrightarrow{r?(i,t,\tilde{v},\ell')}$, $\mathrm{comp}(w, i)$, $\ell' \in \mathrm{range}_{tl}(\ell)$, and

$$M' \equiv (M_{1,1}'|(M_{1,2}'|\ldots|(M_{i,k}'|M^*)\ldots))\backslash T.$$

Since $M_1 \approx M_2$, $M_2 \stackrel{\overline{b}(w,t,\tilde{v},tl,\ell)}{\Longrightarrow} M_2' \approx M_1'$. Furthermore, $M_2|M \equiv S$, where $S \stackrel{\mathrm{def}}{=} ((\ldots((M_2|M_{1,1})|M_{1,2})|\ldots|M_{i,k})|M^*)\backslash T$ and since

$$S \stackrel{\alpha}{\Longrightarrow} ((\ldots((M_2'|M_{1,1}')|M_{1,2}')|\ldots|M_{i,k}')|M_2)\backslash T \equiv M_2'|M',$$

we conclude that $(M_1'|M', M_2'|M') \in \mathcal{R}$ as required. The case where the roles of M_1 and M are swapped is similar.

This completes the proof. □

It is also worth pointing out that we may establish the following interesting property of mobile nodes pertaining to their ubiquitous nature, namely:

Lemma 2.3 For any process P with identifier id, locations ℓ and ℓ' and $N, H \subseteq \mathcal{L}$, if $(\ell, \ell) \in Nb^+$, where Nb^+ is the transitive closure of relation Nb, then

$$P[\![id, \ell, N, H]\!] \approx P[\![id, \ell', N, H]\!].$$

PROOF: The proof involves showing that

$$\mathcal{S} = \{(P[\![id, \ell, N, H]\!], P[\![id, \ell', N, H]\!])| \text{ for all } P, id, \ell, \ell', N, H\} \cup Id$$

is a bisimulation relation where Id is the identity relation on networks. This is easy to establish by noting that $P[\![id, \ell, N, H]\!] \Longrightarrow P[\![id, \ell', N, H]\!]$ by application of rule (Move) to achieve the migration between locations ℓ and ℓ'. □

Typically, bisimulation relations are used to establish that a system satisfies its specification by describing the two as process-calculus processes and discovering a bisimulation that relates them. Their theory has been developed into two directions. On the one hand, sound and complete axiom systems have been developed for establishing algebraically the equivalence of processes. On the other hand, proof techniques that ease the task of showing two processes to be equivalent have been proposed. The results presented in the next section belong to this latter type.

2.4 Confluence

In this section we develop the notion of *confluence* in our calculus. We recall that a process is *confluent* if, from each of its reachable states, "of any two possible actions, the occurrence of one will never preclude the other" [9]. As shown in [9, 8] for pure CCS, and generalized in other calculi (e.g. [3, 19, 4, 12, 15, 13]), 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. In particular, for a certain class of confluent processes, in order to check that a property is satisfied in every execution of the system it suffices to show that it is satisfied by a single (arbitrary) execution.

We now turn to consider the notion of confluence in our calculus. We begin with the notion of determinacy which makes use of the following notation: given actions α and α' we write $\alpha \bowtie \alpha'$ if α and α' differ only in their specified location, i.e. $\alpha' = \alpha[\ell'/\ell]$, for some ℓ and ℓ'.

Definition 2.4 M is *determinate* if, for every derivative M' of M and for all actions α, α', $\alpha \bowtie \alpha'$, whenever $M' \xrightarrow{\alpha} M_1$ and $M' \stackrel{\hat{\alpha'}}{\Longrightarrow} M_2$ then $M_1 \approx M_2$.

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. This is irrespective of the location at which the action is taking place which explains the use of the $\bowtie$ operator. As an example, consider processes

$$\begin{aligned} M_1 &\stackrel{\text{def}}{=} \overline{b}(-, t, \tilde{v}, \mathrm{n}).\mathbf{0}{:}[\![1, \ell, \{2\}, \{2\}]\!] \\ M_2 &\stackrel{\text{def}}{=} (r(t, \tilde{x}).\overline{b}(-, s, \tilde{x}, \mathrm{h}).\mathbf{0},){:}[\![2, \ell, \{1\}, \{1\}]\!] \end{aligned}$$

We observe that M_1 is determinate, whereas $M_1 \mid M_2$ is not: Assuming that $\ell \notin \mathrm{range}_n(\ell')$,

$$\begin{aligned} M_1 \mid M_2 &\xrightarrow{\overline{b}(-,t,\tilde{v},\mathrm{n},\ell)} M \stackrel{\text{def}}{=} \mathbf{0}{:}[\![1, \ell, \ldots]\!] \mid \overline{b}(-, s, \tilde{x}, \mathrm{h}).\mathbf{0}{:}[\![2, \ell, \ldots]\!] \\ M_1 \mid M_2 &\xrightarrow{\mu} \xrightarrow{\overline{b}(-,t,\tilde{v},\mathrm{n},\ell')} M' \stackrel{\text{def}}{=} \mathbf{0}{:}[\![1, \ell', \ldots]\!] \mid (r(t, \tilde{x}).\overline{b}(-, s, \tilde{x}, \mathrm{h}).\mathbf{0}){:}[\![2, \ell, \ldots]\!] \end{aligned}$$

and $M \not\approx M'$.

As in pure CCS, a CSDT process bisimilar to a determinate process is determinate, and determinate processes are bisimilar if they may perform the same sequence of visible actions. The following lemma summarizes conditions under which determinacy is preserved in CSDT networks.

Lemma 2.5

1. If $P{:}\sigma$ is determinate then so is $(\eta.P){:}\sigma$.
2. If $P_1{:}\sigma + \ldots + P_n{:}\sigma$ is determinate then so is $\mathsf{cond}\ (e_1 \rhd P_1, \ldots, e_n \rhd P_n){:}\sigma$.

Proof: The proof follows easily and it employs Lemma 2.3. □

In the case of (2), we point out that the determinacy of each of the $P_i{:}\sigma$ is not sufficient for the determinacy of $\mathsf{cond}\ (e_1 \rhd P_1, \ldots, e_n \rhd P_n){:}\sigma$. As a counter-example, consider the process

$$S \stackrel{\text{def}}{=} \mathsf{cond}\ (2 \in N \rhd \overline{b}(-, t, 1, \mathrm{h}).\mathbf{0}, 2 \notin N \rhd \overline{b}(-, t, 1, \mathrm{h}).\overline{b}(-, t, 1, \mathrm{h}).\mathbf{0})[\![1, \ell, \{2\}, \emptyset]\!]$$

where $2 \notin \text{range}_{\text{n}}(\ell)$. This is due to the fact that, while in S the first case of the condition is enabled, it is possible that $S \xrightarrow{\mu} (...)[\![1, \ell, \emptyset, \emptyset]\!]$, and, thereby, the second case of the conditional is enabled.

We point out that determinacy is not preserved by parallel composition. This is illustrated by the composition $M_1 \mid M_2$ discussed above. Notably, determinacy is also not preserved by restriction. For instance, if we take

$$M \stackrel{\text{def}}{=} (\overline{b}(-, t, \tilde{u}, \text{n}).M_1 + \overline{b}(-, t, \tilde{v}, \text{n}).M_2) : [\![id, \ell, N, H]\!]$$

with M_1 and M_2 determinate networks with $t \notin \text{types}(M_1) \cup \text{types}(M_2)$ such that $M_1 \not\approx M_2$, we have that M is determinate but $M \backslash \{t\}$ is not: $M \backslash \{t\} \xrightarrow{\tau} M_1 \backslash \{t\}$, $M \backslash \{t\} \xrightarrow{\tau} M_2 \backslash \{t\}$ and $M_1 \backslash \{t\} \not\approx M_2 \backslash \{t\}$.

In order to strengthen determinacy into a notion preserved by a wider range of operators, and, in particular, parallel composition Milner [9] introduced the notion of confluence. According to the definition of [9], a CCS process P is *confluent* if it is determinate and, for each of its derivatives Q and distinct actions α, β, given the transitions to Q_1 and Q_2, the following diagram can be completed.

$$\begin{array}{ccc} Q & \xrightarrow{\alpha} & Q_1 \\ {\scriptstyle\beta}\ \Downarrow & & {\scriptstyle\hat{\beta}}\ \Downarrow \\ Q_2 & \xRightarrow{\hat{\alpha}} & Q_2' \sim Q_1' \end{array}$$

The study of confluence was extended and generalized in various other calculi. It is interesting to note that, in value-passing calculi, a restriction was imposed on the above definition so that α and β are not inputs on the same channel, thus, allowing to consider a process such as $P \stackrel{\text{def}}{=} a(x).\overline{b}(x).\mathbf{0}$ to be confluent, despite that fact that transitions such as $P \xrightarrow{a(2)} \overline{b}(2).\mathbf{0}$ and $P \xrightarrow{a(3)} \overline{b}(3).\mathbf{0}$ cannot be 'brought together' in order to complete the diagram above. Despite this fact, it appears natural to classify P as a confluent process. Indeed, investigation of confluence in the context of value-passing calculi resulted in extending the CCS definition above to take account of substitution of values [17, 19]. The definitions highlight the asymmetry between input and output actions by considering them separately.

In the context of our work, we also choose to treat emissions and receipts of messages differently. In particular, regarding the receipt of messages we note that receipt of messages of ad hoc network nodes is continuously enabled and the receipt of a certain message does not preclude the receipt of another. It is however often the case that at some point certain messages are not/no longer relevant for a computation and are discarded without affecting a node's execution. We take this into account as follows:

Definition 2.6 M is *confluent* if it is determinate and, for each of its derivatives M' and distinct actions α, β, where additionally $\neg(\alpha \bowtie \beta)$, whenever $M' \xrightarrow{\alpha} M_1$ and $M' \xRightarrow{\beta} M_2$ then,

1. if $\alpha = r(id, t, \tilde{u}, \ell)$ and $\beta = r(id, t, \tilde{v}, \ell')$, then, either (1) $M_1 \approx M_2$, or (2) $M_1 \xRightarrow{\beta} M_1' \approx M_2$, or (3) $M_2 \xRightarrow{\alpha} M_2' \approx M_1$, or (4) $M_1 \xRightarrow{\beta} M_1'$, $M_2 \xRightarrow{\alpha} M_2'$, and $M_1' \approx M_2'$,
2. otherwise, 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 point out that in the first clause of the definition, case (1) refers to the case that both messages are in fact nonsignificant to the process, cases (2) and (3) refer to the case

where only one of the two messages is relevant and case (4) to the case when receipt of the one does not preclude receipt of the other.

We may see that bisimilarity preserves confluence. Furthermore, confluent processes possess an interesting property regarding internal actions. We define a process P to be *τ-inert* if, for each derivative Q of P, if $Q \stackrel{\tau}{\longrightarrow} Q'$ or $Q \stackrel{\mu}{\longrightarrow} Q'$, then $Q \approx Q'$. By a generalization of the proof in CCS, we obtain:

Lemma 2.7 If M is confluent then M is τ-inert.

We proceed with a result on the preservation of confluence by CSDT operators.

Lemma 2.8

1. if $P{:}\sigma$ is confluent then so are $(\tau.P){:}\sigma$ and $(\overline{b}(w,t,\tilde{x},tl).P){:}\sigma$.
2. If $P_1{:}\sigma + \ldots + P_n{:}\sigma$ is confluent, then so is $\mathsf{cond}\ (e_1 \rhd P_1, \ldots, e_n \rhd P_n){:}\sigma$.
3. If M is confluent, then so is $M\backslash T$.

We note that the cases of the input prefix and parallel composition are missing from our result. Regarding input prefix, it is not difficult to see that, in general, the diamond property of confluence need not be completed for distinct inputs. Of course, this does not imply that no input-prefixed process is confluent. For instance, consider process $C\langle A\rangle : \sigma$, where

$$\begin{array}{lcl} C\langle A\rangle & \stackrel{\text{def}}{=} & r(t,x).C\langle A - \{t,x\}\rangle,\ A \neq \emptyset \\ C\langle A\rangle & \stackrel{\text{def}}{=} & \mathbf{0},\ \text{if } A = \emptyset \end{array}$$

This process is confluent since any two inputs can be brought together to the same A-derivative. Indeed, this is the intended interpretation for confluence in our study and in fact this is the type of processes that we deal with in our case study.

In the case of $\eta.P$ where η is an output action, it is interesting to note that the confluence diagram can be completed despite of node mobility. For example, assuming that $\mathrm{l}(\sigma) = \ell$, although $M \stackrel{\text{def}}{=} [\overline{b}(w,t,\tilde{x},tl).P]{:}\sigma$ can perform actions $M \stackrel{\overline{b}(w,t,\tilde{x},tl,\ell)}{\longrightarrow} M_1 = P{:}\sigma$ and

$$M \stackrel{\mu}{\longrightarrow} [\overline{b}(w,t,\tilde{x},tl).P)]{:}\sigma[\ell'/\ell] \stackrel{\overline{b}(w,t,\tilde{x},tl,\ell')}{\longrightarrow} M_2 = P{:}\sigma[\ell'/\ell],$$

we have $\overline{b}(w,\ell,t,\tilde{x}) \bowtie \overline{b}(w,\ell',t,\tilde{x})$ thus the confluence of the process is not endangered.

For the case of parallel composition, if we take

$$\begin{array}{lcl} M_1 & = & (r(t,x).\overline{b}(-,\mathit{success},x,\mathrm{n}).P){:}[\![1,\ell,\{2\},\{2\}]\!] \\ M_2 & = & (\overline{b}(-,t,\tilde{v},\mathrm{n}).\mathbf{0}){:}[\![2,\ell,\{1\},\{1\}]\!], \end{array}$$

we have that although M_1 and M_2 are both confluent, their composition is not: Assuming that $\ell \notin \mathrm{range}_{\mathrm{n}}(\ell')$, we have

$$\begin{array}{lll} M_1|M_2 & \stackrel{\overline{b}(-,\mathit{success},\tilde{v},\mathrm{n},\ell)}{\longrightarrow} & M = \overline{b}(-,\mathit{success},\tilde{v},\mathrm{n}).P){:}[\![1,\ell,\{2\},\{2\}]\!]|\mathbf{0}{:}[\![2,\ell,\{1\},\{1\}]\!] \\ M_1|M_2 & \stackrel{\mu}{\longrightarrow} \stackrel{\overline{b}(-,\mathit{success},\tilde{v},\mathrm{n},\ell')}{\longrightarrow} & M' = M_1|\mathbf{0}{:}[\![2,\ell',\{1\},\{1\}]\!] \end{array}$$

and clearly $M \not\approx M'$.

2.5 Bisimulation and Confluence in Stationary Systems

Our discussion in the previous section has illustrated that the main obstacle in establishing the compositionality of confluence with respect to parallel composition, as well as other operators, is that of node mobility. Since during the verification of ad hoc network systems correctness criteria focus on the behavior of systems once they become stable for a sufficient amount of time, in what follows we develop a theory of partial confluence that focuses on system behavior not involving the use of the μ action. This restriction gives rise to notions of bisimulation, determinacy and confluence in stationary systems and allows for providing a compositional theory of confluence.

We begin by defining a notion of weak transition, which we call a *stationary weak transition*, that permits τ actions and excludes μ actions: Given an action α we write $P \Longrightarrow_s Q$ for $P(\xrightarrow{\tau})^*Q$, $P \stackrel{\alpha}{\Longrightarrow}_s Q$ for $P \Longrightarrow_s \xrightarrow{\alpha} \Longrightarrow_s Q$, and $P \stackrel{\hat{\alpha}}{\Longrightarrow}_s Q$ for $P \Longrightarrow_s Q$ if $\alpha = \tau$, $P \stackrel{r(id,\ell,t,\tilde{v})}{\Longrightarrow}_s Q$, if $\alpha = r?(id, \ell, t, \tilde{v})$, and $P \stackrel{\alpha}{\Longrightarrow}_s Q$ otherwise.

In a similar vein, the new notion of bisimilarity, S-bisimilarity matches the behavior of equivalent systems excluding μ-actions:

Definition 2.9 *S-Bisimilarity* is the largest symmetric relation, denoted by $\approx_s$, such that, if $P \approx_s Q$ and $P \xrightarrow{\alpha} P'$, $\alpha \in Act - \{\mu\}$, there exists Q' such that $Q \stackrel{\hat{\alpha}}{\Longrightarrow}_s Q'$ and $P' \approx_s Q'$.

It is easy to see that $\approx \subset \approx_s$. We may prove that:

Lemma 2.10 S-bisimilarity is a congruence relation.

PROOF: The proof follows similarly to that of Lemma 2.2. □

Based on the notion of S-confluence, we may define the notion of S-determinacy as follows:

Definition 2.11 M is *S-determinate* if, for every derivative M' of M and for all actions $\alpha \in Act - \{\mu\}$, whenever $M' \xrightarrow{\alpha} M_1$ and $M' \stackrel{\alpha}{\Longrightarrow}_s M_2$ then $M_1 \approx_s M_2$.

Lemma 2.12

1. If $P{:}\sigma$ is S-determinate then so is $(\eta.P){:}\sigma$.
2. If $P_i{:}\sigma$, $1 \leq i \leq n$, are S-determinate then so is $\mathsf{cond}\ (e_1 \rhd P_1, \ldots, e_n \rhd P_n){:}\sigma$.
3. If M is S-determinate then so is $M \backslash F$.

PROOF: The proof follows similarly to the case of Lemma 2.5. □

As in the case of determinacy, S-determinacy is not preserved by parallel composition. To ameliorate this fact we define S-confluence as follows.

Definition 2.13 M is *S-confluent* if it is S-determinate and, for each of its derivatives M' and distinct actions α, β, whenever $M' \xrightarrow{\alpha} M_1$ and $M' \stackrel{\beta}{\Longrightarrow}_s M_2$ then,

1. if $\alpha = r(id, t, \tilde{u}, \ell)$ and $\beta = r(id, t, \tilde{v}, \ell)$, then, either (1) $M_1 \approx_s M_2$, or (2) $M_1 \stackrel{\beta}{\Longrightarrow}_s M_1' \approx_s M_2$, or (3) $M_2 \stackrel{\alpha}{\Longrightarrow}_s M_2' \approx_s M_1$, or (4) $M_1 \stackrel{\beta}{\Longrightarrow}_s M_1'$, $M_2 \stackrel{\alpha}{\Longrightarrow}_s M_2'$, and $M_1' \approx_s M_2'$,
2. otherwise, there are M_1' and M_2' such that $M_2 \stackrel{\hat{\alpha}}{\Longrightarrow}_s M_2'$, $M_1 \stackrel{\hat{\beta}}{\Longrightarrow}_s M_1'$ and $M_1' \approx_s M_2'$

We may see that S-bisimilarity preserves S-confluence. Furthermore, S-confluent processes possess the property that their observable behavior remains unaffected by τ actions. In particular, we define a process P to be τ_s-*inert* if, for each derivative Q of P, if $Q \xrightarrow{\tau} Q'$, then $Q \approx_s Q'$. We may prove:

Lemma 2.14 If M is S-confluent then M is τ_s-inert.

We proceed with a result on the preservation of confluence by CSDT operators.

Lemma 2.15

1. if $P{:}\sigma$ is S-confluent then so are $(\tau.P){:}\sigma$ and $(\bar{b}(w,t,\tilde{x},tl).P){:}\sigma$.
2. If $P_i{:}\sigma$, $1 \leq i \leq n$, are S-confluent, then so is $\mathsf{cond}\ (e_1 \rhd P_1, \ldots, e_n \rhd P_n){:}\sigma$.
3. If M is S-confluent, then so is $M\backslash T$.
4. If M_1 and M_2 are S-confluent then so is $M_1 \mid M_2$.

PROOF: We may show that any derivative M' of M is confluent by a case analysis on the possible actions of M', similarly to [15]. □

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 [20]. 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. Note that in what follows we use the term "neighbors" of a node to refer to the nodes that fall within its normal transmission range, and the term "heartbeat neighbors" for the nodes that fall within its high transmission range.

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 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 so-called *heartbeat* messages to the environment, which are essentially messages sent at a high transmission level. The absence of a heartbeat message from its leader triggers a node to initiate a new computation of a leader. Note that such an absence may just indicate that the node is outside of the high transmission range of the leader even though they belong to the same connected component of the network. However, this does not affect the correctness of the algorithm. Based on this extension, computation 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 meet each other, 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, each adopts the leader with the largest identifier. An exception to this is the case when the two nodes that have come into contact have the same previous-leader field (a piece of information maintained in the mobile version of the algorithm), in which case they proceed as with the static case with the highest valued-computation being the winner.

3.2 Specification of the Protocol

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

- **election**: messages of this type are invitations by a node to another to join its computation.
- **ack1**: a message of this 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 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. Such messages are important for the following reason: If a node x departs from its location, enters a new computation and returns to its previous location before its initial neighbors notice its departure, these reply messages will provide these initial neighbors that x is no longer part of their computation and thus they will no longer expect x to reply.
- **hbeat**: a message of this type is emitted by a leader node.

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).

$$\mathrm{S} \stackrel{\text{def}}{=} (\prod_{k \in K} \mathrm{Elected}\langle b_k, s_k, lead_k \rangle{:}\sigma_k) \backslash T,$$

where $T = \{\mathbf{election}, \mathbf{ack0}, \mathbf{ack1}, \mathbf{leader}, \mathbf{reply}\}$. Thus, we restrict all but **hbeat** messages emitted by leader nodes which are the messages we wish to observe. Based on these messages we will subsequently express the correctness criterion of the algorithm.

The description of the behavior of nodes can be found in Figure 1. We now proceed to provide a step-by-step description of the specification.

A node in Elected possesses a leader, *lead*, a source, *src*, and a status, b, which records whether the process needs to broadcast its leader. A node finds itself in this mode if it possesses a leader who it believes to be available. It can perform the following actions:

1. If it has not done so already, that is $b = 0$, it may broadcast its leader via a **leader** message, in which case b becomes equal to 1. If the node is a leader (i.e. $id = lead$) it sends a heartbeat to notify its heartbeat neighbors of its availability. On the other hand, if it observes the absence of its leader (condition $lead \notin \mathrm{H}$), it enters InComp mode, wherein it begins a quest for a new leader with source its own identifier.

2. It may receive a **leader** broadcast from one of its neighbors. If the advertised leader is greater than the node's leader, it adopts it and it sets $b = 0$. If not, it ignores the message.

3. It may emit a message of type **reply** and thus announce to its neighbors that it participates in a computation with source s.

4. It may receive an **election** message which amounts to an invitation to enter a new computation to elect a new leader. It accepts such a message only if the leader announced in the invitation coincides with its own leader. Thus, it interprets this message as the loss of its leader, a fact first realized by the source existing in the message. In response, it enters the InComp mode with father the node who sent the invitation and source the source provided in the invitation.

The InComp process has a number of parameters: c records whether the node needs to broadcast a leader election invitation to its neighbors. The second parameter is the node's father to which eventually an **ack1** message needs to be returned (unless the node itself is the root of the tree). *src* and *lead* are the computation's source and previous leader, whereas *max* is the maximum identifier observed by the node, initially set as the node's own identifier. Sets R and A 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 sets are regularly updated according to the node's interface: we write $R' = R \cap \mathrm{N}$, $A' = A \cap \mathrm{N}$ and $father' = father$, if $father \in \mathrm{N}$, and $NULL$, otherwise. It is worth noting that at these points (as well as others, e.g. "$father = \mathrm{id}$", "$lead \in \mathrm{H}$"), the ability of referring to the node's interface plays a crucial role for the specification of the protocol. Furthermore, the fact that sets N and H are in fact the believed sets of neighbors and may contain errors is realistic and it allows us to explore the correctness of the protocol in its full generality.

An InComp node behaves as follows:

1. If it has not done so already, that is $c = 0$, it may emit an **election** message to its neighbors thus inviting them to enter its computation for a leader.

$\text{Elected}\langle 0, src, lead\rangle \stackrel{\text{def}}{=} \overline{b}(-, \textbf{leader}, \langle lead\rangle, \text{n}).\, \text{Elected}\langle 1, src, lead\rangle$

$\text{Elected}\langle 1, src, lead\rangle \stackrel{\text{def}}{=}$
$\quad \textsf{cond}\ (lead = \text{id} \rhd \overline{b}(-, \textbf{hbeat}, \langle lead\rangle, \text{h}).\text{Elected}\langle 1, src, lead\rangle,$
$\qquad lead \notin \text{H} \rhd \text{InComp}\langle 0, \text{id}, \text{id}, \text{N}, \emptyset, \text{id}, lead\rangle)$
$\quad + r(\textbf{leader}, \langle lead'\rangle).\ \textsf{cond}\ (lead < lead' \rhd \text{Elected}\langle 0, src, lead'\rangle,$
$\qquad\qquad true \rhd \text{Elected}\langle 1, src, lead\rangle)$
$\quad + \overline{b}(-, \textbf{reply}, \langle \text{id}, s\rangle, \text{n}).\, \text{Elected}\langle 1, src, lead\rangle$
$\quad + r(\textbf{election}, \langle j, l, s\rangle).\ \textsf{cond}\ (l = lead \rhd \text{InComp}\langle 0, j, s, \text{N} - \{j\}, \emptyset, \text{id}, lead\rangle,$
$\qquad\qquad true \rhd \text{Elected}\langle 1, src, lead\rangle)$

$\text{InComp}\langle c, NULL, src, R, A, max, lead\rangle \stackrel{\text{def}}{=} \text{InComp}\langle c, \text{id}, src, R', A', max, lead\rangle$
$\text{InComp}\langle 1, father, src, \emptyset, \emptyset, max, lead\rangle \stackrel{\text{def}}{=}$
$\quad \textsf{cond}\ (father = \text{id} \rhd \text{Elected}\langle 0, src, max\rangle,$
$\qquad true \rhd \overline{b}(father, \textbf{ack1}, \langle \text{id}, scr, max\rangle, \text{n}).\, \text{Leader}\langle father, src, max, lead\rangle)$
$\text{InComp}\langle c, father, src, R, A, max, lead\rangle \stackrel{\text{def}}{=}$
$\quad \textsf{cond}\ (c = 0 \rhd \overline{b}(-, \textbf{election}, \langle \text{id}, lead, scr\rangle, \text{n}).\, \text{InComp}\langle 1, father', scr, R', A', max, lead\rangle)$
$\quad + \sum_{j \in A} \overline{b}(j, \textbf{ack0}, \langle \text{id}\rangle, \text{n}).\, \text{InComp}\langle 1, father', scr, R', A' - \{j\}, max, lead\rangle$
$\quad + r(\textbf{election}, \langle j, l, s\rangle).$
$\qquad \textsf{cond}\ (l = lead \wedge s > src \rhd \text{InComp}\langle 0, j, s, \text{N} - \{j\}, \emptyset, max, lead\rangle,$
$\qquad\quad l = lead \wedge s = scr \rhd \text{InComp}\langle c, father', scr, R', A' \cup \{j\}, max, lead\rangle,$
$\qquad\quad true \rhd \text{InComp}\langle c, father, scr, R', A', max, lead\rangle)$
$\quad + r(\textbf{ack0}, \langle j\rangle).\, \text{InComp}\langle 1, father', src, R' - \{j\}, A', max, lead\rangle$
$\quad + r(\textbf{ack1}, \langle j, s, max'\rangle).$
$\qquad \textsf{cond}\ (s = src \wedge max' > max \rhd \text{InComp}\langle c, father', src, R' - \{j\}, A', max', lead\rangle,$
$\qquad\quad s = src \wedge max' \leq max \rhd \text{InComp}\langle c, father', src, R' - \{j\}, A', max, lead\rangle,$
$\qquad\quad true \rhd \text{InComp}\langle c, father', src, R' - \{j\}, A', max, lead\rangle)$
$\quad + r(\textbf{leader}, \langle l\rangle).\, \text{InComp}\langle c, father', src, R', A', max, lead\rangle$
$\quad + r(\textbf{reply}, \langle j, s\rangle).$
$\qquad \textsf{cond}\ (src \neq s \rhd \text{InComp}\langle c, father', src, R' - \{j\}, A' - \{j\}, max, lead\rangle,$
$\qquad\quad true \rhd \text{InComp}\langle c, father', src, R', A', max, lead\rangle)$
$\quad + \overline{b}(-, \textbf{reply}, \langle \text{id}, src\rangle, \text{n}).\, \text{InComp}\langle c, father', src, R', A', max, lead\rangle$

$\text{Leader}\langle NULL, src, max, lead\rangle \stackrel{\text{def}}{=} \text{Elected}\langle 0, src, max_i\rangle$
$\text{Leader}\langle father, src, max, lead\rangle \stackrel{\text{def}}{=}$
$\quad r(\textbf{election}, \langle j, l, s\rangle).$
$\qquad \textsf{cond}\ (l = lead \wedge s > src \rhd \text{InComp}\langle 0, j, s, \text{N} - \{j\}, \emptyset, max, lead\rangle,$
$\qquad\quad true \rhd \text{Leader}\langle father', src, max, lead\rangle)$
$\quad + r(\textbf{leader}, \langle l\rangle).\, \text{Elected}\langle 0, src, l\rangle$
$\quad + \overline{b}(-, \textbf{reply}, \langle \text{id}, src\rangle, \text{n}).\, \text{Leader}\langle father', scr, max, lead\rangle$

Figure 2: The description of a node

2. It may send an **ack0** message to inform a neighbor in set A that it will not enter its election computation. Note that set A is initialized to 0 and it obtains members as a result of **election** requests arriving at a node, as is explained below.

3. A node may receive an **election** message. If this originates from a neighbor j with the same *lead* parameter (i.e. a node that was in the same neighborhood during the last election and not a newly-arrived node) and a higher *src* parameter, then it accepts the request and re-starts computation with an appropriate initialization of its fields. Note that j becomes the father of the node. If the message was received from a neighbor

with the same $lead$ and src parameters, that is a node in the same computation, it stores j in the set A so as to inform j that it does not accept it as its father. If both of the above fail, the node ignores the message.

4. A node may receive an **ack0** message from a neighbor who does not accept it as its father. As a result it removes the identifier of the neighbor from its set R.

5. A node may receive an **ack1** message from a neighbor. Such a message contains the maximum value of the subtree rooted at the sender. The receiving node accepts this message only if originates from a neighbor with the same source (i.e. the same computation) and, it adopts the received max' value if this is greater than its own max.

6. A node may receive **leader** announcements from its neighbors which it ignores.

7. A node may receive **reply** messages from its neighbors. If it observes that a neighbor is involved in a computation with a different source it removes it from both its R and A sets.

8. A node broadcasts to its neighbors a reply message containing its source.

9. If at any point during execution it loses its father then it sets itself as its father (thus becoming the root of the tree under construction with respect to the leader election taking place in its downwards nodes).

10. If a node has completed its computation, that is, its pending acknowledgments A have been sent out and its children R have responded, then it sends an **ack1** message to its father containing the maximum identifier of its subtree and enters Leader therein waiting to hear the announcement of the leader, or, if it is the root of its tree, it enters Elected with leader its max parameter.

Finally, node Leader$\langle father, src, max, lead\rangle$ awaits to be notified of the component's leader by its father $father$. It recalls its computation characterized by its source and previous leader (src and $lead$) as well as the maximum node it has observed from its downstream nodes, max. It behaves as follows:

1. It accepts an **election** message only if it originates from a node with the same leader and a higher source.

2. It accepts **leader** messages and, in fact, it adopts the first leader that becomes available through such a message.

3. It announces its own status by broadcasting its source via **reply** messages.

4. In case of the loss of its father, the node enters the Elected with leader its value max.

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 of the network, where connectedness is defined in terms of neighborhood according to the normal transmission range, 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 and we prove the following:

Theorem 3.1 *$S_1 \approx_s (\prod_{k\in K} \mathrm{Elected}\langle 1, s, max_k\rangle{:}\sigma_k)\backslash T$ where max_k is the maximum node identifier in the connected component of node k.*

In words, our correctness criterion states that, if at some point nodes stop moving, all nodes will learn a leader which is the node with the maximum identifier within their connected component. It is similar in spirit to the criterion of [20], where a temporal logic proof is presented.

Moving on with the proof of the theorem, let us consider an arbitrary derivative of S. This has the form:

$$\begin{aligned} S_1 \;=\; & (\prod_{i\in E} \mathrm{Elected}\langle b_i, s_i, lead_i\rangle{:}\sigma_i \mid \prod_{i\in L} \mathrm{Leader}\langle f_i, src_i, max_i, lead_i\rangle{:}\sigma_i \\ & \mid \prod_{i\in C} \mathrm{InComp}\langle c_i, f_i, src_i, R_i, A_i, max_i, lead_i\rangle{:}\sigma_i)\backslash T \end{aligned}$$

We can prove by induction on the length of the computation $S \Longrightarrow_s S_1$ that:

- For all i, $lead_i, max_i \geq \mathrm{id}(\sigma_i)$;
- If $j \in R_i$ and $src_j = src_i$, then $j \in C$ and $i \in \{f_j\} \cup A_j$.

Furthermore, we assume that for all i, j, if $i \in \mathrm{N}_j$ then $i \in \mathrm{range_n}(j)$, and if $i \in \mathrm{H}_j$ then $i \in \mathrm{range_h}(j)$.

Recall that K is the set of network nodes, so, $K = E \cup L \cup C$. 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 $S_1 = (\prod_{g\in G} CC_g)\backslash T$, where:

$$\begin{aligned} CC_g \;=\; & (\prod_{i\in E_g} \mathrm{Elected}\langle b_i, s_i, lead_i\rangle{:}\sigma_i \mid \prod_{i\in L_g} \mathrm{Leader}\langle f_i, src_i, max_i, lead_i\rangle{:}\sigma_i \\ & \mid \prod_{i\in C_g} \mathrm{InComp}\langle c_i, f_i, src_i, R_i, A_i, max_i, lead_i\rangle{:}\sigma_i)\backslash T \end{aligned}$$

We begin to consider computation in each of the neighborhoods N_g, $g \in G$, of the network and we establish that each such neighborhood will choose as its leader the maximum identifier known by the nodes:

Lemma 3.2 *$CC_g \approx_s Spec_1$, where $Spec_1 \stackrel{\mathrm{def}}{=} (\prod_{i\in N_g} \mathrm{Elected}\langle 1, s_i, max_g\rangle{:}\sigma_i)\backslash T$ and $max_g = max\{max_i | i \in N_g\}$.*

The proof of this result has a similar flow 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 forest as opposed to a tree, which is the case in the static

case. The second point is that the nodes comprising CC_g are not themselves S-confluent, therefore we cannot conclude the S-confluence of CC_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 CC_g, F_g, where each node x in F_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 2.15, F_g is an S-confluent process and we exhibit a stationary weak transition which leads to $Spec_1$. By F_g's S-confluence and the fact that $F_g \Longrightarrow_s Spec_1$, we conclude that $F_g \approx_s Spec_1$. Here we may observe the power of (S-)confluence: it is sufficient to observe only a single execution of our system. Then, confluence guarantees that in fact all possible executions lead to the same state up-to bisimulation and, then, by τ_s-inertness the result follows. For the second step of the proof we show that whenever $CC_g \Longrightarrow_s D$ where exists a F_g (i.e. an assignment of fathers to nodes) that is similar to D. Since this is true for any D we conclude that CC_g cannot diverge from the behavior of the F_g's and also that it is destined to produce only their behavior. Hence, we may deduce that in fact $CC_g \approx_s Spec_1$ as required.

The restricted type of systems employed in the first phase of the proof use the following processes:

$$
\begin{aligned}
&\mathrm{Elected}'\langle 1, src, f, src', lead\rangle \stackrel{\text{def}}{=}\\
&\quad \ldots\\
&\quad + r(\mathbf{election}, \langle j, l, s\rangle).\,\mathsf{cond}\ ((l = lead \wedge j = f \wedge s = src' \wedge s \neq \mathrm{id}) \rhd\\
&\qquad\qquad \mathrm{InComp}'\langle 0, j, s, j, s, \mathrm{N} - \{j\}, \emptyset, \mathrm{id}, lead\rangle\\
&\qquad true \rhd \mathrm{Elected}'\langle i, src, f, src', lead\rangle)\\
&\mathrm{InComp}'\langle c, father, src, f, src', R, A, max, lead\rangle \stackrel{\text{def}}{=}\\
&\quad \ldots\\
&\quad + r(\mathbf{election}, \langle j, l, s\rangle).\\
&\qquad \mathsf{cond}\ (l = lead \wedge src \neq src' \wedge j = f \wedge s = src'\\
&\qquad\qquad \rhd \mathrm{InComp}'\langle 0, j, s, j, s, \mathrm{N} - \{j\}, \emptyset, max, lead\rangle\\
&\qquad\quad l = lead \rhd\\
&\qquad\qquad \mathrm{InComp}'\langle c, father', scr, f, src', R' - \{j\}, A, max, lead\rangle\\
&\qquad\quad true \rhd \mathrm{InComp}'\langle c, father', scr, f, src', R', A', max, lead\rangle)\\
&\quad + \ldots\\
&\mathrm{Leader}'\langle father, src, f, src', max, lead\rangle \stackrel{\text{def}}{=}\\
&\quad r(\mathbf{election}, \langle j, l, s\rangle).\\
&\qquad \mathsf{cond}\ (l = lead \wedge src \neq src' \wedge j = f \wedge s = src'\\
&\qquad\qquad \rhd \mathrm{InComp}'\langle 0, j, s, j, s, \mathrm{N} - \{j\}, es, max, lead\rangle\\
&\qquad\quad true \rhd \mathrm{Leader}'\langle father', scr, f, src', max, lead\rangle)\\
&\quad + \ldots
\end{aligned}
$$

Thus, Elected$'$ is similar to Elected except that it may only be activated by a signal from a specified node-source pair, (f, src') unless $src' = \mathrm{id}$ in which case it may only initiate a computation. Similarly, InComp$'$ and Leader$'$ are similar to InComp and Leader, respectively, except that they may only be activated by a signal from a specified node-source pair, (f, src') unless they are already in the specific computation.

Returning to a component CC_g, we observe that it may contain a number of active computations which will eventually merge into one. To begin with, nodes that have the same leader and are within reach of each other will join into one computation tree and elect the maximum available identifier as their leader. Subsequently, and after they reach the state Elected they will start flooding their leader within the component with the overall maximum identifier surviving the process as the component's leader. To capture the forest

created at the first phase of the computation (note that there may be many of them) let us write $L = \{lead_i | i \in N_g\}$ and K_l, $l \in L$, for the subset of N_g consisting of all nodes with leader parameter l. Each K_l can be partitioned into a set of connected components which we will denote by K_l^m, $m \in M_l$. We point out that although the nodes of N_g are connected to each other, it is possible that subsets of N_g with the same *lead* parameter are not connected to each other (nodes with a different *lead* parameter may be lying in between). We now define a set of agents $\mathcal{F}$ that capture these forests of computation that may arise within a CC_g, ranged over by F_g:

$$\begin{aligned} F_g \stackrel{\text{def}}{=} \prod_{l\in L}\prod_{m\in M_l}\Big(&\prod_{i\in E_{l,m}} \text{Elected}'\langle b_i, src_i, f_i, \mathbf{s}_{l,m}, l\rangle{:}\sigma_i \\ &| \prod_{i\in L_{l,m}} \text{Leader}'\langle father_i, src_i, f_i, \mathbf{s}_{l,m}, max_i, lead_i\rangle{:}\sigma_i \\ &| \prod_{i\in C_{l,m}} \text{InComp}'\langle c_i, father_i, src_i, f_i, \mathbf{s}_{l,m}, R_i, A_i, max_i, lead_i\rangle{:}\sigma_i\Big)\backslash T \end{aligned}$$

where if $mx = max\{src_i | i \in L_{l,m} \cup C_{l,m}\}$, then $\mathbf{s}_{l,m} \in \{mx\} \cup \{\text{id}_i | i \in E_{l,m}, i > mx_s\}$, and for all $i \neq \mathbf{s}_{l,m}$, $f_i \in \text{N}_i$ and $\{(\text{id}_i, f_i) | i \in K_l, -\{\mathbf{s}_{l,m}\}\}$ is a spanning tree of the component rooted at node $\mathbf{s}_{l,m}$.

We begin by showing:

Lemma 3.3 $F_g \Longrightarrow_s Spec_1$.

Proof. Let $l \in L$, $m \in M_l$, and $D_{l,m}$ be the maximum distance of a node in K_l^m from the root $\mathbf{s}_{l,m}$ of the spanning tree and let $D = max_{l,m} D_{l,m}$. Fix sets $A_{l,m}^d, 0 \leq d \leq D_{l,m}$, such that:

$$A_{l,m}^d = \begin{cases} \{\mathbf{s}_{l,m}\} & d = 0 \\ \{i \in K_l^m | f_i \in A_{l,m}^{d-1}\} & d > 0 \end{cases}$$

In other words, $A_{l,m}^1$ contains the nodes that have $\mathbf{s}_{l,m}$ as their father, $A_{l,m}^2$ the nodes whose father is a node of $A_{l,m}^2$, and so on. We are particularly interested in subsets of these sets containing those nodes that have not yet entered a computation initiated by $\mathbf{s}_{l,m}$. So let us write $B_{l,m}^0$ for the nodes with identifier $\mathbf{s}_{l,m}$, that is, the future roots of the forest of the network, that are either in state Leader or in state $\text{InComp}\langle 0, \ldots\rangle$. Similarly, let us write $B_{l,m}^i$ for the subset of $A_{l,m}^i$ containing those nodes that do not have $\mathbf{s}_{l,m}$ as their source. Further, let us write $Ch_i = \{j \mid f_j = i\}$ and F^d, $0 \leq d \leq D$ for the process

$$\begin{aligned} F^d \stackrel{\text{def}}{=} \prod_{l\in L}\prod_{m\in M_l}\Big(&\prod_{i\in E_{l,m}-(B_{l,m}^0\cup B_{l,m}^d)} \text{Elected}'\langle b_i, src_i, f_i, \mathbf{s}_{l,m}, l\rangle{:}\sigma_i \\ &| \prod_{i\in L_{l,m}-(B_{l,m}^0\cup B_{l,m}^d)} \text{Leader}'\langle father_i, src_i, f_i, \mathbf{s}_{l,m}, max_i, lead_i\rangle{:}\sigma_i \\ &| \prod_{i\in C_{l,m}-(B_{l,m}^0\cup B_{l,m}^d)} \text{InComp}'\langle c_i, father_i, src_i, f_i, \mathbf{s}_{l,m}, R_i, A_i, max_i, lead_i\rangle{:}\sigma_i \\ &| \prod_{i\in (B_{l,m}^0\cup B_{l,m}^{d-1})} \text{InComp}'\langle 1, f_i, \mathbf{s}_{l,m}, f_i, \mathbf{s}_{l,m}, \text{N}_i - \{f_i\}, \emptyset, max_i, lead_i\rangle{:}\sigma_i \\ &| \prod_{i\in B_{l,m}^d} \text{InComp}'\langle 0, f_i, \mathbf{s}_{l,m}, f_i, \mathbf{s}_{l,m}, \text{N}_i - \{f_i\}, \emptyset, max_i, lead_i\rangle{:}\sigma_i\Big)\backslash T \end{aligned}$$

We may see that

$$F_g = F^0 \Longrightarrow_s F^1 \Longrightarrow_s \ldots \Longrightarrow_s F^D\,.$$

The first move $F^0 \Longrightarrow_s F^1$ concerns the emission of an election request by all nodes in $B^0_{l,m}$. As a result of this broadcast all these nodes will enter state $\mathrm{InComp}'\langle 1, \ldots\rangle$. Simultaneously, all nodes at broadcasting distance from these nodes, will accept the invitation and become $\mathrm{InComp}'\langle 0, \mathbf{s}_{l,m}, \mathbf{s}_{l,m}, \ldots\rangle$, as required. Similarly, at subsequent moves, nodes at distance i will forward the invitations to their neighbors, leading the computation to F^D.

At this point, all pending **leader** requests can be sent and **ack0** acknowledgements can be returned, and any **reply** messages can be emitted yielding $F^D \Longrightarrow_s$

$$\begin{aligned} G^D \stackrel{\text{def}}{=} \prod_{l\in L}\prod_{m\in M_l} (&\mathrm{InComp}'\langle 1, \mathbf{s}_{l,m}, \mathbf{s}_{l,m}, \mathbf{s}_{l,m}, \mathbf{s}_{l,m}, Ch_i, \emptyset, max_i, lead_i\rangle{:}\sigma_i \\ &\mid \prod_{i\in L_{l,k}-\{\mathbf{s}_{l,m}\}} \mathrm{InComp}'\langle 1, f_i, \mathbf{s}_{l,m}, f_i, \mathbf{s}_{l,m}, Ch_i, \emptyset, max_i, lead_i\rangle{:}\sigma_i)\backslash T \end{aligned}$$

Now, let us write G^d, $1 \le d \le D$, for the process

$$\begin{aligned} G^d \stackrel{\text{def}}{=} \prod_{l\in L}\prod_{m\in M_l} (&\mathrm{InComp}'\langle 1, \mathbf{s}_{l,m}, \mathbf{s}_{l,m}, \mathbf{s}_{l,m}, \mathbf{s}_{l,m}, Ch_i, \emptyset, max_i, lead_i\rangle{:}\sigma \\ &\mid \prod_{i\in A^d_{l,m}} \mathrm{InComp}'\langle 1, f_i, \mathbf{s}_{l,m}, f_i, \mathbf{s}_{l,m}, \emptyset, \emptyset, max_i, lead_i\rangle{:}\sigma_i \\ &\mid \prod_{i\in A^{d+1}_{l,m}\cup\ldots\cup A^{D_{l,m}}_{l,m}} \mathrm{LeaderMode}'\langle f_i, s_i, f_i, s_i, mx_i, lead_i\rangle)\backslash T \end{aligned}$$

where mx_i is the maximum max-identifier of all nodes in the subtree rooted at node i. It is easy to see that

$$G^D \Longrightarrow_s G^{D-1} \Longrightarrow_s \ldots \Longrightarrow_s G^1\,.$$

In particular, for any $0 \le d < D$, $G^d \Longrightarrow_s G^{d-1}$ consists of the emission of all **ack1**-messages by all nodes in $A^d_{l,m}$. Since $\cup_{l,m} A^d_{l,m}$ contains exactly the children of all nodes in $\cup_{l,m} A^{d-1}_{l,m}$ we may confirm the move.

Now, similarly, $G^1 \Longrightarrow_s G^0$, where

$$\begin{aligned} G^0 \stackrel{\text{def}}{=} \prod_{l\in L}\prod_{m\in M_l} (&\mathrm{Elected}'\langle 0, \mathbf{s}_{l,m}, \mathbf{s}_{l,m}, \mathbf{s}_{l,m}, lead_{l,m}\rangle{:}\sigma \\ &\mid \prod_{i\in K_{l,m}-\{\mathbf{s}_{l,m}\}} \mathrm{LeaderMode}'\langle f_i, s_i, f_i, s_i, mx_i, lead_i\rangle)\backslash T \end{aligned}$$

where $lead_{l,m}$ is the maximum max-identifier existing in $K_{l,m}$. It is now trivial to see that flooding of leader messages will result in the adoption of the maximum $lead_{l,m}$ by all nodes in the component, so that

$$G^0 \Longrightarrow_s Spec_1$$

as required.

□

We may now observe that F_g is an S-confluent process:

Lemma 3.4 *F_g is S-confluent.*

Proof. We may check that processes InComp$'$, Leader$'$ and Elected$'$, are S-confluent by construction. Thus, by Theorem 2.15 the result follows. □

Given the S-confluence of F_g and Lemma 2.14 we conclude that:

Corollary 3.5 *For all $F_g \in \mathcal{F}_g$, $F_g \approx_s Spec_1$.*

Having used confluence to analyze the behavior of F_g, we can now relate it to that of CC_g. Let P range over derivatives of CC_g and T range over derivatives of F_g. First, we introduce a notion of *similarity* between derivatives of F_g and CC_g. We say that P and T are *similar* if the computation sources nodes in T are eligible source nodes in P and additionally, the set of nodes in P that have this source form a subtree of the spanning tree of T. All such nodes are in the same state in both P and T with the exception of InComp and Leader nodes of P that are not part of a $\mathbf{s}_{l,m}$ computation: in T these nodes are awaiting to be awoken from their father in order to enter the computation. The precise definition is as follows:

Definition 3.6 *Let*

$$P \stackrel{\text{def}}{=} \prod_{l\in L}\prod_{m\in M_l}\Big(\prod_{i\in E_{l,m}} \text{Elected}\langle b_i, s_i, l\rangle{:}\sigma_i \,\Big|\, \prod_{i\in L_{l,m}} \text{Leader}\langle f_i, src_i, max_i, l\rangle{:}\sigma_i \,\Big|\, \prod_{i\in C_{l,m}} \text{InComp}\langle c_i, f_i, src_i, R_i, A_i, max_i, l\rangle{:}\sigma_i\Big)\backslash T$$

and

$$\begin{aligned} T \stackrel{\text{def}}{=} \prod_{l\in L}\prod_{m\in M_l}\Big(&\prod_{i\in E^1_{l,m}} \text{Elected}'\langle b_i, s_i, f_i, \boldsymbol{s}_{l,m}, l\rangle{:}\sigma_i \\ &\Big|\prod_{i\in L_{l,m}} \text{Leader}'\langle father_i, src_i, f_i, \boldsymbol{s}_{l,m}, max_i, l\rangle{:}\sigma_i \\ &\Big|\prod_{i\in C^1_{l,m}} \text{InComp}'\langle c_i, f_i, \boldsymbol{s}_{l,m}, f_i, \boldsymbol{s}_{l,m}, R_i, A_i, max_i, l\rangle{:}\sigma_i \\ &\Big|\prod_{i\in C^2_{l,m}\cup E^2_{l,m}} \text{InComp}'\langle 0, f_i, \boldsymbol{s}_{l,m}, f_i, \boldsymbol{s}_{l,m}, \text{N}_i, \emptyset, max_i, l\rangle{:}\sigma_i\Big)\backslash T \end{aligned}$$

where

- *if $mx_s = max\{src_i | i \in L_{l,k} \cup C_{l,k}\}$, then $\boldsymbol{s}_{l,m} \in \{mx_s\} \cup \{\text{id}_i | i \in E_{l,k}, i > mx_s\}$,*
- *for all $i \neq \boldsymbol{s}_{l,m}$, $f_i \in \text{N}_i$ and $\{(id_i, f_i) | i \in K_{l,k} - \{C^1_{l,m}\}$ is a spanning tree of the component rooted at node $\boldsymbol{s}_{l,m}$,*
- *$E^1_{l,m} = \{i \in E_{l,m} | s_i = \boldsymbol{s}_{l,m}\}$ and $E^1_{l,m} = \{i \in E_{l,m} | s_i \neq \boldsymbol{s}_{l,m}\}$, and*
- *$C^1_{l,m} = \{i \in C_{l,m} | s_i = \boldsymbol{s}_{l,m}\}$ and $C^1_{l,m} = \{i \in C_{l,m} | s_i \neq \boldsymbol{s}_{l,m}\}$.*

Then we say that P and T are similar *processes.*

Lemma 3.7 *$\mathcal{R} = \{\langle T, P\rangle | P$ and T are similar$\}$ is a strong simulation.*

Proof. An observation of each of the nodes makes it obvious that any action a node can perform within T, it can also perform it within P. This leads us to the conclusion that any $T \longrightarrow T'$ can be mimicked by $P \longrightarrow P'$ with T' and P' similar processes. □

Our next result establishes a correspondence between CC_g and agents $F_g \in \mathcal{F}_g$.

Lemma 3.8 *If $CC_g \stackrel{w}{\Longrightarrow}_s P$ then there exists F_g such that, $F_g \stackrel{w}{\Longrightarrow}_s T$ and P and T are similar.*

Proof. Suppose

$$CC_g \stackrel{\text{def}}{=} \prod_{l\in L}\prod_{m\in M_l}\Big(\prod_{i\in E_{l,m}} \text{Elected}\langle b_i, s_i, l\rangle{:}\sigma_i \Big| \prod_{i\in L_{l,m}} \text{Leader}\langle f_i, src_i, max_i, l\rangle{:}\sigma_i \Big| \prod_{i\in C_{l,m}} \text{InComp}\langle c_i, f_i, src_i, R_i, A_i, max_i, l\rangle{:}\sigma_i\Big)\backslash T$$

and

$$P \stackrel{\text{def}}{=} \prod_{l\in L}\prod_{m\in M_l}\Big(\prod_{i\in E'_{l,m}} \text{Elected}\langle b_i, s_i, l\rangle{:}\sigma_i \Big| \prod_{i\in L'_{l,m}} \text{Leader}\langle f_i, src_i, max_i, l\rangle{:}\sigma_i \Big| \prod_{i\in C'_{l,m}} \text{InComp}\langle c_i, f_i, src_i, R_i, A_i, max_i, l\rangle{:}\sigma_i\Big)\backslash T$$

where $CC_g \stackrel{w}{\Longrightarrow}_s P$. We say that $F_g \in \mathcal{F}$, is *compatible* with the computation, if

$$\begin{aligned} F_g \stackrel{\text{def}}{=} \prod_{l\in L}\prod_{m\in M_l}\Big(&\prod_{i\in E^1_{l,m}} \text{Elected}'\langle b_i, s_i, f_i, \mathbf{s}_{l,m}, l\rangle{:}\sigma_i \\ &\Big| \prod_{i\in L_{l,m}} \text{Leader}'\langle father_i, src_i, f_i, \mathbf{s}_{l,m}, max_i, l\rangle{:}\sigma_i \\ &\Big| \prod_{i\in C^1_{l,m}} \text{InComp}'\langle c_i, f_i, \mathbf{s}_{l,m}, f_i, \mathbf{s}_{l,m}, R_i, A_i, max_i, l\rangle{:}\sigma_i \\ &\Big| \prod_{i\in C^2_{l,m}\cup E^2_{l,m}} \text{InComp}'\langle 0, f_i, \mathbf{s}_{l,m}, f_i, \mathbf{s}_{l,m}, \mathrm{N}_i, \emptyset, max_i, l\rangle{:}\sigma_i\Big)\backslash T \end{aligned}$$

where F_g is similar to CC_g and, if $mx_s = max\{src_i | i \in L'_{l,k} \cup C'_{l,k}\}$, then $\mathbf{s}_{l,m} \in \{mx_s\} \cup \{\text{id}_i | i \in E'_{l,k}, i > mx_s\}$.

We will prove the result by induction on the length, n, of the transition $CC_g \stackrel{w}{\Longrightarrow}_s P$. The base case $n = 0$ is trivially true for any $F_g \in \mathcal{F}_g$. Suppose that the result holds for $n = k - 1$ and consider $CC_g \stackrel{w}{\Longrightarrow}_s P' \stackrel{\alpha}{\longrightarrow} P$ a transition of length k. Let F_g be compatible with the computation. Then, F_g is also compatible with the computation $CC_g \stackrel{w}{\Longrightarrow}_s P'$ and, by the induction hypothesis, $F_g \stackrel{w}{\Longrightarrow}_s T'$ where P' and T' are similar. Now, consider the transition $P' \stackrel{\alpha}{\longrightarrow} P$. The following cases exist:

- $\alpha = \tau$ and the internal action does not involve a source $\mathbf{s}_{l,m}$. Then, we may see that for $T = T'$, $T' \stackrel{\epsilon}{\Longrightarrow}_s T'$ with P and T being similar.
- $\alpha = \tau$ and the internal action involves a source $\mathbf{s}_{l,m}$. Then, using a case analysis similar to the one found in the proof of Lemma 3.7, we may find appropriate T such that $T' \stackrel{\tau}{\longrightarrow} T$ and T, P similar.

- $\alpha = \overline{b}(-, \mathbf{hbeat}, lead_i, \mathrm{h})$. Then there must exist a process $\mathrm{Elected}(1, s_i, lead_i)$ in P' and $P = P'$. But then, the same process must exist in T' and clearly $T' \xrightarrow{\alpha} T'$. Since P' and T' are similar this completes the case.

□

We can now prove our main theorem. We have seen that $CC_g \Longrightarrow_s Spec_1$. Further, suppose that $P_0 \stackrel{\alpha}{\Longrightarrow}_s$ with $\alpha \neq \overline{leader}(\mathbf{max})$. Then, there exists T_0 such that $T_0 \stackrel{\alpha}{\Longrightarrow}_s$. However, this is in conflict with Corollary 3.5. Finally, for the same reason, it is not possible that $P_0 \Longrightarrow_s P_1' \not\longrightarrow$. This implies that $P_0 \approx_s T_0$, as required.

We prove the following result:

Lemma 3.9 *$CC_g \approx_s Spec_1$.*

Proof: Consider the relation

$$\begin{aligned}\mathcal{R} \;=\; \{(P,S) \mid\; & CC_g \xrightarrow{w} P,\ \exists F_g \in \mathcal{F}_g \cdot F_g \xrightarrow{w} T \text{ where } P, T \text{ are similar, and} \\ & Spec \stackrel{w}{\Longrightarrow}_s S \approx_s T\}\end{aligned}$$

We may prove that $\mathcal{R}$ is a weak bisimulation. Let $(P,S) \in \mathcal{R}$ and:

- Suppose $P \xrightarrow{\alpha} P'$, then, by Lemma 3.8 there exists $F_g \in \mathcal{F}_g$ such that $F_g \xrightarrow{w} T \xrightarrow{\alpha} T'$ where P' is similar to T'. Since $F_g \approx_s Spec_1$, $S \approx_s T$, there exists T'' such that $F_g \xrightarrow{w} T''$ and $T'' \approx_s S$. By the S-confluence of F_g we have that $T \approx_s T''$ and $T \approx_s S$. This implies that $S \stackrel{\alpha}{\Longrightarrow}_s S' \approx_s T'$, as required.
- Suppose $S \xrightarrow{\alpha} S'$.Then, $T \xrightarrow{\alpha} T'$ where $S' \approx_s T'$. Since P simulates T, $P \stackrel{\alpha}{\Longrightarrow}_s P'$ where P' simulates T', as required

This implies that $\mathcal{R}$ is a weak bisimulation and, since $CC_g, Spec_1) \in \mathcal{R}$, this completes the proof. □

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

Lemma 3.10 *$S_1 \approx_s S_2$, where $S_2 \stackrel{\text{def}}{=} (\prod_{g \in G} \prod_{i \in N_g} \mathrm{Elected}\langle 1, s_i, max_g\rangle{:}\sigma_i)\backslash T$ and $max_g = max\{max_i | i \in N_g\}$.*

Now two cases exist: Either the max_g are nodes located in the neighborhoods CC_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 $max = \mathrm{id}$, we conclude that:

Lemma 3.11 *$S_2 \approx_s S_3$, where $S_3 \stackrel{\text{def}}{=} (\prod_{g \in G} \prod_{i \in N_g} \mathrm{Elected}\langle 1, s_i, max_g\rangle{:}\sigma_i)\backslash T$ where $max_g = max\{\mathrm{id}_i | i \in N_g\}$.*

This gives us the correctness of Theorem 3 and completes the proof. □

4 Conclusions

In this paper we have introduced a process calculus for reasoning about systems with dynamic interconnections and their protocols. A salient aspect of the calculus is its ability to encode a protocol that manages topology (e.g. discovery of neighbors) thus enabling one to

focus on the details of an algorithm at hand while abstracting from topology computations. The change in the physical topology is discovered by the semantics non-deterministically via internal actions in a way similar to the implementation of the movement of nodes that cause these topology changes.

Communication in our calculus takes place via a broadcast-style of messages which also implements point-to-point communication. This type of communication is used to model the broadcasting of messages emitted for an intended recipient which often takes place in real networks. Again here, we may imagine the existence of a lower-level protocol at each node that filters among the received messages those intended for the node. Furthermore, we have allowed nodes to broadcast their messages at two different transmission levels. As illustrated via our case study, the ability to do so is not only useful with respect to saving power but can also play an important role for protocol correctness. We point out that this could easily be extended to a wider range of transmission levels by considering a set of transmission levels Tl and replacing sets N and H in a node's interface by a relation $\mathcal{I} \times Tl$. Here we opt to implement two levels for the sake of simplicity and because this is already sufficient for our case study. Finally, we have introduced the notion of a message type and a hiding operator based on types. These message types are reminiscent of tags by which various applications prefix different messages according to their roles and the restriction operator then allows an observer to focus on a part of the message exchange. As illustrated via our case study, this can be especially useful for analysis purposes and specifically for expressing appropriate correctness criteria via bisimulations. In our case study, restriction over types allowed us to focus on leader announcements emitted by leader nodes.

We have also developed a theory of confluence for our process calculus. This theory is fairly simple when compared to similar theories for e.g. the π-calculus: the absence of channels for communication and the broadcasting style of message transmission has removed a number of considerations relating to confluence preservation. We believe that the results can be extended to other calculi featuring a broadcasting style of communication. The results we developed proved to be useful for reasoning about the leader-election algorithm under study. They allowed us to conclude the confluence of the analyzed systems merely by construction and then to deduce the desired bisimilarity via τ-inertness. This was a significant aid for the proof since it permitted us to observe the system in a single execution path which we selected based on our understanding of the algorithm. By confluence, we then deduced that all executions lead to the same state up-to bisimulation.

We have illustrated the applicability of the new formalism via the analysis of a leader-election MANET protocol. In [20], the authors also give a proof of their algorithm using temporal logic: in particular they show a "weak form of stabilization of the algorithm", namely, that after a finite number of topological changes, the algorithm converges to a desired stable state in a finite amount of time. As we do in our proof, they operate under the assumption of no message loss. The same algorithm has also been considered in [18] where its correctness was analyzed for a number of tree and ring-structured initial topologies for networks with 5 to 8 nodes. In particular, it was shown automatically that it is possible that eventually a node with the maximum id in a connected component will be elected as the leader of the component and that every node connected to it via one or more hops will learn about its election. The reachability nature of this result is due to the lossy communication implemented in the ω-calculus resulting in no guarantees that a leader will be elected.

In conclusion, we believe that CSDT can be applied for specifying and verifying a wide range of dynamic-topology protocols and that the theory of confluence can play an important role in facilitating the construction of their proofs. This belief is supported by our current work on specifying and verifying a MANET routing algorithm. In future work

we are planning to extend our framework in the presence of message loss.

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 in 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] X. Liu and D. Walker. Confluence of processes and systems of objects. In *Proceedings of TAPSOFT'95*, LNCS 915, pages 217–231, 1995.

[5] M. Merro. An observational theory for mobile ad hoc networks. *Information and Computation*, 207(2):194–208, 2009.

[6] M. Merro and E. Sibilio. A timed calculus for wireless systems. In *Proceedings of FSEN'09, Revised Selected Papers*, LNCS 5961, pages 228–243, 2010.

[7] N. Mezzetti and D. Sangiorgi. Towards a calculus for wireless systems. *Electronic Notes in Theoretical Computer Science*, 158:331–353, 2006.

[8] R. Milner. *A Calculus of Communicating Systems*. Springer, 1980.

[9] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[10] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts 1 and 2. *Information and Computation*, 100:1–77, 1992.

[11] S. Nanz and C. Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1-2):203–227, 2006.

[12] U. Nestmann. *On Determinacy and Non-determinacy in Concurrent Programming*. PhD thesis, University of Erlangen, 1996.

[13] A. Philippou and G. Michael. Verification techniques for distributed algorithms. In *Proceedings of OPODIS'06*, LNCS 4305, pages 172–186, 2006.

[14] A. Philippou and D. Walker. On transformations of concurrent object programs. In *Proceedings of CONCUR'96*, LNCS 1119, pages 131–146, 1996.

[15] A. Philippou and D. Walker. On confluence in the π-calculus. In *Proceedings of ICALP'97*, LNCS 1256, pages 314–324, 1997.

[16] K. V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25(2-3):285–327, 1995.

[17] M. Sanderson. *Proof Techniques for CCS*. PhD thesis, University of Edinburgh, 1982.

[18] 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.

[19] C. Tofts. *Proof Methods and Pragmatics for Parallel Programming*. PhD thesis, University of Edinburgh, 1990.

[20] 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.