A Process Calculus for Mobile Ad Hoc Networks

Anu Singh, C. R. Ramakrishnan, and Scott A. Smolka

Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-4400, USA
E-mail: {anusingh,cram,sas}@cs.sunysb.edu

Abstract. We present the ω-calculus, a process calculus for formally modeling and reasoning about *Mobile Ad Hoc Wireless Networks* (MANETs) and their protocols. The ω-calculus naturally captures essential characteristics of MANETs, including the ability of a MANET node to broadcast a message to any other node within its physical transmission range (and no others), and to move in and out of the transmission range of other nodes in the network. A key feature of the ω-calculus is the separation of a node's communication and computational behavior, described by an ω-process, from the description of its physical transmission range, referred to as an ω-process *interface*.

Our main technical results are as follows. We give a formal operational semantics of the ω-calculus in terms of labeled transition systems and show that the state reachability problem is decidable for finite-control ω-processes. We also prove that the ω-calculus is a conservative extension of the π-calculus, and that late bisimulation (appropriately lifted from the π-calculus to the ω-calculus) is a congruence. Congruence results are also established for a weak version of late bisimulation, which abstracts away from two types of internal actions: τ-actions, as in the π-calculus, and μ-actions, signaling node movement. Finally, we illustrate the practical utility of the calculus by developing and analyzing a formal model of a leader-election protocol for MANETs.

1 Introduction

A Mobile Ad Hoc Network (MANET) is a network of autonomous mobile nodes connected by wireless links. Each node N has a physical transmission range within which it can directly transmit data to other nodes. Any node that falls within N's transmission range is considered a *neighbor* of N. Nodes can move freely in a MANET, leading to rapid change in the network's communication topology.

Two aspects of MANETs make them especially difficult to model using existing formal specification languages such as process algebras. First, MANETs use wireless links for local broadcast communication: a MANET node can transmit a message simultaneously to all nodes within its transmission range, but the message cannot be received by any node outside that range. Secondly, the neighborhood of nodes that lie within the transmission range of a node can change unpredictably due to node movement, thereby altering the set of nodes that can receive a transmitted message.

Ideally, the specification of a MANET node's control behavior should be independent of its neighborhood information. Since, however, the eventual recipients of a local broadcast message depend on this information, a model of a MANET-based protocol given in a traditional process calculus must intermix the computation of neighborhood information with the protocol's control behavior. This tends to render such models unnatural and unnecessarily complex.

In this paper, we present the ω-calculus, a conservative extension of the π-calculus that has been designed expressly to address the MANET modeling problems outlined above. A key feature of the ω-calculus is the separation of a node's communication and computational behavior, described by an ω-process, from the description of its physical transmission range, referred to as an ω-process *interface*. This separation allows one to model the control behavior of a MANET protocol, using ω-processes, independently from the protocol's underlying communication topology, using process interfaces. (A similar separation of concerns has been achieved in several recently introduced process calculi for wireless and mobile networks [5, 8, 9, 12], but not, as we argue in Section 6, as simply and naturally as in the ω-calculus.)

As discussed further in Section 2, ω-process interfaces are comprised of *groups*, which operationally function as local broadcast ports. Mobility is captured in the ω-calculus via the dynamic creation of new groups and dynamically changing process interfaces. The group-based abstraction for local broadcast in a wireless network is a natural one; it appears also in [6], where it is shown how to model MANETs in the UPPAAL model checker for timed automata.

Main Contributions. The rest of the paper is organized around our main technical results, which include the following:

- Section 2 provides an informal introduction to the basic features of the ω-calculus.
- Section 3 presents the formal operational semantics of the ω-calculus in terms of labeled transition systems and structural-congruence rules. The calculus is presented in three stages: ω_0, the core version of the calculus, focuses on local broadcast and mobility; ω_1 extends ω_0 with unicast communication and scope extrusion; ω_2 extends ω_1 by allowing multi-threaded behavior at the process level. Unless otherwise noted, the expression "the ω-calculus" refers to ω_2, the most general version of the calculus. We in fact show in Section 4 that ω_2 is a conservative extension of the π-calculus.
- Section 4 defines bisimulation for the ω-calculus and proves that it is a congruence. We obtain similar results for a weak version of bisimulation, which treats as unobservable two types of internal actions: τ-actions, as in the π-calculus, and μ-actions, signaling node movement. Full proofs of these results appear in [16].
- Section 5 illustrates the practical utility of the calculus by developing and analyzing a formal ω-calculus model of a leader-election algorithm for MANETs [17].

Section 6 considers related work and Section 7 offers our concluding remarks.

2 The ω-Calculus: An Informal Introduction

As an illustrative example of the ω-calculus, consider the MANET of Fig. 1(a) comprising the four nodes N_1, N_2, N_3, N_4. The dotted circle centered around a node indicates the node's transmission range, and all nodes are assumed to have the same transmission range. Thus, N_1 is within the transmission range of N_2, N_3, and N_4 and vice versa, and N_2 and N_4 are in each other's transmission range. Fig. 1(b) highlights the *maximal sets of neighboring nodes* in the network, one covering N_1, N_2, and N_4, and the other covering N_1 and N_3. A maximal set of neighboring nodes corresponds to a *maximal clique* in the network's node connectivity graph (Fig. 1(c)), and, equivalently, to an ω-calculus *group* (local broadcast port), as illustrated in Fig. 1(d). The set of groups

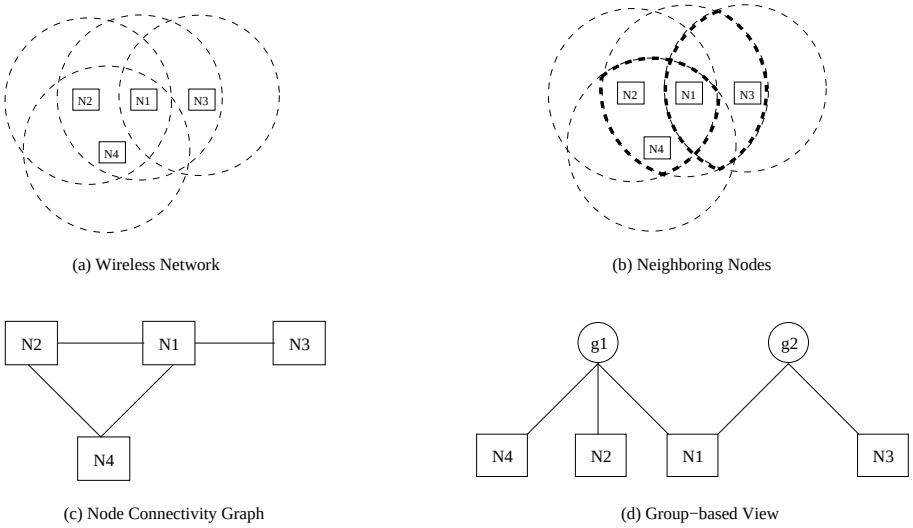

Fig. 1. Multiple views of a MANET network.

to which a node is connected is specified by the *interface* of the underlying process; i.e. the process executing at the node. Thus, the ω-calculus expression for the network is the parallel composition $N_1|N_2|N_3|N_4$, where $N_1 = P_1 : \{g_1, g_2\}$, $N_2 = P_2 : \{g_1\}$, $N_3 = P_3 : \{g_2\}$, $N_4 = P_4 : \{g_1\}$, for process expressions P_1, P_2, P_3 and P_4.

Note that process interfaces may contain groups that do not correspond to maximal cliques. Such groups are redundant in the sense that do not represent any additional connectivity information. Group g_2 of Fig. 2 is an example of a redundant group. A *canonical* form for ω-calculus expressions can be defined in which redundant groups are elided.

Fig. 1 provides multiple views of the topology of the MANET at a particular moment in time. As discussed below, the network topology may change over time due to node movement, a feature of MANETs captured operationally in the ω-calculus via dynamic updates of process interfaces.

Local Broadcast in the ω-calculus. The ω-calculus action to locally broadcast a value x is $\overline{\mathbf{b}}x$, while $\mathbf{r}(y)$ is the action for receiving a value y. Thus, when a process transmits a message, only the message x to be sent is included in the specification. The set of possible recipients depends on the process's current interface: only those processes that share a common group with the sender can receive the message and this information is not part of the syntax of local broadcast actions. In the example of Fig. 1, if P_2 can broadcast a message and P_1, P_3, P_4 are willing to receive it, then the expression

$$N = \mathbf{r}(x).P_1' : \{g_1, g_2\} \mid \overline{\mathbf{b}}u.P_2' : \{g_1\} \mid \mathbf{r}(y).P_3' : \{g_2\} \mid \mathbf{r}(z).P_4' : \{g_1\}$$

may evolve to

$$N = P_1'\{u/x\} : \{g_1, g_2\} \mid P_2' : \{g_1\} \mid \mathbf{r}(y).P_3' : \{g_2\} \mid P_4'\{u/z\} : \{g_1\}$$

Observe that P_3 does not receive the message since N_3 is not in N_2's neighborhood.

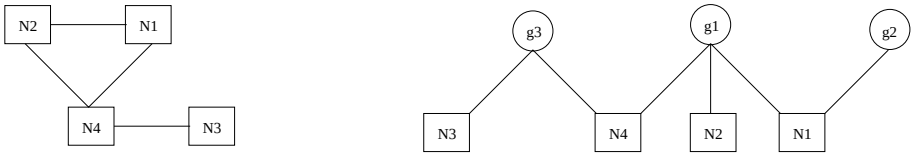

Fig. 2. (a) Node Connectivity Graph after N_3's movement and (b) View in ω-calculus.

Node mobility in the ω-calculus. Node mobility is captured through the dynamic creation of new groups and dynamically changing process interfaces. Fig. 2 shows the topology of the network of Fig. 1 after N_3 moves away from N_1's transmission range and into N_4's transmission range. N_3's movement means that the ω-calculus expression

$$(\nu g_1)(\nu g_2)(P_1 : \{g_1, g_2\} \mid P_2 : \{g_1\} \mid P_3 : \{g_2\} \mid P_4 : \{g_1\})$$

evolves to

$$(\nu g_1)(\nu g_2)(P_1 : \{g_1, g_2\} \mid P_2 : \{g_1\} \mid (\nu g_3)(P_3 : \{g_3\} \mid P_4 : \{g_1, g_3\}))$$

The new group g_3 in the above expression represents the new maximal set of neighboring nodes N_3 and N_4 that arises post-movement. We use the familiar νg notation for group-name scoping.

Nodes vs. Processes. In an ω-calculus specification, nodes typically represent physical devices; as such, the calculus does not provide a primitive for node creation. Process creation, however, is supported, as processes model programs and other executables that execute within the confines of a device.

3 Syntax and Transitional Semantics of the ω-Calculus

We begin this section by presenting the syntax and semantics of ω_0, our core calculus for MANETs. We then introduce the extensions to ω_0 that result in the more expressive ω_1- and ω_2-calculi.

3.1 Syntax of ω_0

A system description in the ω_0-calculus comprises a set of *nodes*, each of which runs a sequential *process* annotated by its *interface*. We use **N** and **P** to denote the sets of all nodes and all processes, respectively, with M, N ranging over nodes and P, Q ranging over processes. We also use names drawn from two disjoint sets: **Pn** and **Gn**. The names in **Pn**, called *pnames* for *process names*, are used for data values. The names in **Gn**, called *gnames* for *group names*, are used for process interfaces. We use x, y, z to range over **Pn** and g (possibly subscripted) to range over **Gn**. The ω_0-calculus has a two-level syntax describing nodes and processes, respectively.

The syntax of ω_0-calculus processes is defined by the following grammar:

$$\begin{aligned} P &::= \mathit{nil} \mid \mathit{Act}.P \mid P + P \mid [x = y]P \mid A(\vec{x}) \\ \mathit{Act} &::= \overline{\mathbf{b}}x \mid \mathbf{r}(x) \mid \tau \end{aligned}$$

N1.	$M \equiv M \mid \mathbf{0}$	N8.	$P{:}G \equiv Q{:}G$, if $P \equiv Q$
N2.	$M_1 \mid M_2 \equiv M_2 \mid M_1$	N9.	$P{:}G \equiv (\nu g)(P{:}G \cup \{g\})$, if $g \notin G$
N3.	$(M_1 \mid M_2) \mid M_3 \equiv M_1 \mid (M_2 \mid M_3)$		
N4.	$(\nu g)M \equiv M$, if $g \notin \mathit{fgn}(M)$		
N5.	$(\nu g)M \mid N \equiv (\nu g)(M \mid N)$, if $g \notin \mathit{fgn}(N)$	P1.	$P + Q \equiv Q + P$
N6.	$(\nu g_1)(\nu g_2)M \equiv (\nu g_2)(\nu g_1)M$	P2.	$(P + Q) + R \equiv P + (Q + R)$
N7.	$M \equiv N$, if $M \equiv_\alpha N$	P3.	$P \equiv Q$, if $P \equiv_\alpha Q$

Table 1. Structural congruence relation.

Action $\overline{\mathbf{b}}x$ represents the local broadcast of a value x, while the reception of a locally broadcasted value is denoted by $\mathbf{r}(x)$. Internal (silent) actions are denoted by τ. Process *nil* is the deadlocked process; $\mathit{Act}.P$ is the process that can perform action *Act* and then behave as P; and $+$ is the operator for nondeterministic choice. Process $[x = y]P$ (where x and y are pnames) behaves as P if names x and y match, and as *nil* otherwise. $A(\vec{x})$ denotes *process invocation*, where A is a process name (having a corresponding definition) and $\vec{x}$ is a comma-separated list of actual parameters (pnames) of the invocation. A process definition is of the form $A(\vec{x}) \stackrel{\text{def}}{=} P$, and associates a process name A and a list of formal parameters $\vec{x}$ (i.e. distinct pnames) with process expression P. Process definitions may be recursive.

The following grammar defines the syntax of ω_0-calculus node expressions:

$$M \ ::= \ \mathbf{0} \ \mid \ P{:}G \ \mid \ (\nu g)M \ \mid \ M|M$$

$\mathbf{0}$ is the inactive node, while $P : G$ is a node with process P having interface (set of gnames) G. The operator (νg) is used to restrict the scopes of gnames. $M|N$ represents the parallel composition of node expressions M and N. Node expressions of the form $P{:}G$ are called *basic node expressions*, while those containing the restriction or parallel operator are called *structured node expressions*. Note that gnames occur only at the node level, capturing the intuition that, in an ad hoc network, the behavioral specification of a (basic) node (represented by its process) is independent of its underlying interface.

Free and Bound Names. Pname x is free in $\overline{\mathbf{b}}x.P$ and bound in $\mathbf{r}(x).P$. Gname g is bound in $(\nu g)M$, and all gnames in G are free in $P : G$. In a process definition of the form $A(\vec{x}) \stackrel{\text{def}}{=} P$, $\vec{x}$ are the only names that may occur free in P. The set of all names, free names and bound names in a process expression P are denoted by $n(P)$, $\mathit{fn}(P)$ and $\mathit{bn}(P)$, respectively. Similarly, the set of all pnames and gnames in a node expression M are denoted by $pn(M)$ and $gn(M)$, and those that occur free are denoted by $\mathit{fpn}(M)$ and $\mathit{fgn}(M)$, respectively. The set of all free names in a node expression M is given by $\mathit{fn}(M) = \mathit{fpn}(M) \cup \mathit{fgn}(M)$. An expression without free names is called *closed*. An expression that is not *closed* is said to be *open*. The theory developed in the following sections is applicable to both *open* and *closed* systems (expressions).

3.2 Transitional Semantics of ω_0

The transitional semantics of the ω_0-calculus is defined in terms of a structural congruence relation $\equiv$ (Table 1) and a labeled transition relation $\stackrel{\alpha}{\longrightarrow} \subseteq \mathbf{N} \times \mathbf{N}$, where α is

Rule Name	Rule	Side Condition
MCAST	$\dfrac{}{(\overline{\mathbf{b}}x.P){:}G \xrightarrow{\overline{G}x} P{:}G}$	
RECV	$\dfrac{}{(\mathbf{r}(x).P){:}G \xrightarrow{G(x)} P{:}G}$	
CHOICE	$\dfrac{P{:}G \xrightarrow{\alpha} P'{:}G}{(P+Q){:}G \xrightarrow{\alpha} P'{:}G}$	
MATCH	$\dfrac{P{:}G \xrightarrow{\alpha} P'{:}G}{([x{=}x]P){:}G \xrightarrow{\alpha} P'{:}G}$	
DEF	$\dfrac{P\{\vec{y}/\vec{x}\}{:}G \xrightarrow{\alpha} P'{:}G}{A(\vec{y}){:}G \xrightarrow{\alpha} P'{:}G}$	$A(\vec{x}) \stackrel{def}{=} P$

Table 2. Transition rules for basic node expressions.

Rule Name	Rule	Side Condition
STRUCT	$\dfrac{N \equiv M \quad M \xrightarrow{\alpha} M' \quad M' \equiv N'}{N \xrightarrow{\alpha} N'}$	
MOBILITY (I)	$\dfrac{}{M \mid P{:}G \xrightarrow{\mu} M \mid P{:}G'}$	$G' \neq G$, $G' \subseteq G \cup \mathit{fgn}(M)$, $\chi(M \mid P{:}G) \models I \implies \chi(M \mid P{:}G') \models I$
PAR	$\dfrac{M \xrightarrow{\alpha} M'}{M \mid N \xrightarrow{\alpha} M' \mid N}$	$\mathit{bn}(\alpha) \cap \mathit{fn}(N) = \emptyset$
COM	$\dfrac{M \xrightarrow{\overline{G}x} M' \quad N \xrightarrow{G'(y)} N'}{M \mid N \xrightarrow{\overline{G}x} M' \mid N'\{x/y\}}$	$G \cap G' \neq \emptyset$
GNAME-RES1	$\dfrac{M \xrightarrow{\alpha} M'}{(\nu\, g)M \xrightarrow{\alpha\setminus\{g\}} (\nu\, g)M'}$	
GNAME-RES2	$\dfrac{M \xrightarrow{\overline{G}x} M'}{(\nu\, g)M \xrightarrow{\tau} (\nu\, g)M'}$	$G = \{g\}$

Table 3. Transition rules for structured node expressions.

the *transition label*. As such, only node expressions have transitions, and these are of the form $M \xrightarrow{\alpha} M'$. There are several varieties of transition labels. When a node of the form $P{:}G$ broadcasts a value x, it generates a transition labeled by $\overline{G}x$. When $P{:}G$ receives a broadcast value x, the corresponding transition label is $G(x)$. Actions μ and

τ also serve as transition labels, with μ, as explained below, indicating node movement, and τ representing internal (silent) actions.

For transition label α, the sets of bound names and gnames of α are denoted $bn(\alpha)$ and $gn(\alpha)$, respectively, and defined as follows:

$bn(\overline{G}x) = \emptyset$, $bn(G(x)) = \{x\}$, $bn(\mu) = \emptyset$, $bn(\tau) = \emptyset$.

$gn(\overline{G}x) = G$, $gn(G(x)) = G$, $gn(\mu) = \emptyset$, $gn(\tau) = \emptyset$.

The transitional semantics of the ω_0-calculus is given by the inference rules of Tables 2 and 3, with the former supplying the inference rules for basic node expressions and the latter for structured node expressions. Rules CHOICE, MATCH, and DEF of Table 2 are standard. Rules MCAST and RECV of Table 2, together with COM of Table 3, define a notion of *local broadcast* communication. RECV states that a basic node with process interface G can receive a local broadcast on any gname in G. This, together with COM, means that a local-broadcast sender can synchronize with any local-broadcast receiver with whom it shares a gname (i.e. the receiver is in the transmission range of the sender).

Local-broadcast synchronization results in a local-broadcast transition label of the form $\overline{G}x$, thereby enabling other receivers to synchronize with the original send action. In contrast to the broadcast calculi of [4, 12], a node that is capable of receiving a local broadcast is not forced to synchronize with the sender. The semantics of local broadcast in the ω-calculus allows a receiver to ignore a local-broadcast event even if this node is in the transmission range of the broadcasting node. A semantics of this nature captures the lossy transmission inherent in MANETs. The semantics of local broadcast can easily be modified to force all potential receivers to receive a local broadcast.

GNAME-RES1 and GNAME-RES2 define the effect of closing the scope of a gname. GNAME-RES1 states that a restricted gname cannot occur in a transition label. In GNAME-RES1, let G be the set of gnames in α; i.e., $G = gn(\alpha)$. Then the transition label $\alpha \setminus \{g\}$ in the consequent of this rule denotes α with the occurrence of gnames in α replaced by $G \setminus \{g\}$, given that $G \setminus \{g\} \neq \emptyset$ and $\alpha \notin \{\tau, \mu\}$. Note that if $\alpha = \tau$ ($\alpha = \mu$), then $\alpha \setminus \{g\} = \tau$ ($\alpha \setminus \{g\} = \mu$). GNAME-RES2 states that when all gnames of a local-broadcast-send action are restricted, it becomes a τ-action. MCAST, GNAME-RES1 and GNAME-RES2 together mean that a local-broadcast send is non-blocking; i.e., it can be performed on a set of restricted groups even when there are no corresponding receive actions. In contrast, other actions containing gnames, such as local-broadcast receive, are not covered by GNAME-RES2, and hence have blocking semantics: a system cannot perform actions involving restricted gnames unless there is a corresponding synchronizing action.

The notion of structural congruence (Table 1) considered in rule STRUCT is defined for processes (rules P1-P3) in the standard way—P and Q are structurally congruent if they are alpha-equivalent or congruent under the associativity and commutativity of the choice ('+') operator—and then lifted to nodes (rules N1-N9). Two basic node expressions are structurally congruent if they have identical process interfaces and run structurally congruent processes (rule N8). Rules N4-N6 are for restriction on gnames. Rule N9 allows basic nodes to create and acquire a new group name or drop a local group name. Structural congruence of nodes includes alpha-equivalence (rule N7) and the associativity and commutativity of the parallel ('|') operator (rules N2 and N3).

Semantics of mobility. The semantics of node movement is defined by the MOBILITY rule, which states that the process interface of node $P:G$ can change from G to G' whenever the node is in parallel with another node M. In particular, the side condition $G' \subseteq G \cup fgn(M)$ stipulates that P may drop gnames from its interface or acquire free gnames from M.

The MOBILITY rule reflects the fact that P's interface may change when node $P : G$, or the nodes around it, are in motion. A change in P's interface may further result in a corresponding change in the overall network topology. Note that the rule does not specify which nodes moved, only that the topology has been updated as the result of movement of one or more nodes.

Process interfaces provide an abstract specification of network topology in terms of node connectivity graphs. Formally, the *node connectivity graph* of a node expression M, denoted by $\chi(M)$, is an undirected graph (V, E) such that V, the set of vertices, are the basic nodes of M (i.e. subexpressions of M of the form $P : G$) and E, the set of edges, is defined as follows. There is an edge between two vertices $P_1:G_1$ and $P_2:G_2$ of $\chi(M)$ only if P_1 and P_2's interfaces overlap; i.e. $G_1 \cap G_2 \not\equiv \emptyset$ (assuming bound names of M are unique and distinct from its free names). The node connectivity graph for the ω_0 node expression of Fig. 1(d) is given in Fig. 1(c).

The third side condition to the MOBILITY rule, expressed in terms of node connectivity graphs, allows one to impose different models of node movement on the calculus. Specifically, the side condition decrees that, whenever $M \xrightarrow{\mu} M'$ is derived using the MOBILITY rule, the resulting transition must preserve a *mobility invariant* expressed as a property over the node connectivity graph. A mobility invariant is a decidable property over undirected graphs. For example, k-connectedness, for a given k, is a candidate mobility invariant, as is `true`, indicating no constraints on node movement. We write $G \models I$ to indicate that undirected graph G possesses property I. We thus have that the MOBILITY rule in particular, and the calculus's semantics in general, are parameterized by the mobility invariant, thus taking into account the constraints on node movement.

3.3 The ω_1- and ω_2-Calculi

The ω_1- and ω_2-calculi are defined in a modular fashion by adding new syntactic constructs, and associated inference rules for their semantics, to the ω_0-calculus.

Extending ω_0 ***to*** ω_1**.** Syntactically, we obtain ω_1 from ω_0 as follows:

- We add restriction operators for *pnames* for both process-level and node-level expressions. We use the standard notation of $(\nu x)P$ for a pname x restricted to a process expression P, and $(\nu x)N$ for a pname x restricted to a node expression N. As usual, x is bound in $(\nu x)P$ and $(\nu x)N$.
- We introduce unicast communication as a prefix operator for process expressions. Although unicast in principle can be implemented on top of broadcast, we prefer to give it first-class status, as it is a frequent action in MANET protocols. Doing so also facilitates concise modeling and deterministic reasoning (only the intended recipient can receive a unicast message). We use the standard notation of $\overline{x}y$ to denote the sending of name y along x, and $x(y)$ to denote the reception of a name

Rule Name	Rule	Side Condition
UNI-SEND	$\dfrac{}{(\overline{z}x.P){:}G \xrightarrow{\overline{z{:}G}x} P{:}G}$	
UNI-RECV	$\dfrac{}{(z(x).P){:}G \xrightarrow{z:G(x)} P{:}G}$	
UNI-COM	$\dfrac{M \xrightarrow{\overline{z:G}x} M' \quad N \xrightarrow{z:G'(y)} N'}{M \mid N \xrightarrow{\tau} M' \mid N'\{x/y\}}$	$G \cap G' \neq \emptyset$

Table 4. Transition rules for unicast communication in ω_1-calculus.

along x that will bind to y. As usual, x and y are free in the expression $\overline{x}y.P$, and x is free and y is bound in $x(y).P$.

Semantically, the introduction of scoped pnames needs new inference rules to handle scope extrusion. We add OPEN and CLOSE rules (as in the π-calculus [11]) and, in addition to the broadcast communication rule (COM) of ω_0, a rule for communication of bound names. We also add RES rules at the process and node levels to disallow communication over a restricted name. These additional rules follow closely the standard rules for handling scopes and scope extrusion in the π-calculus; details are omitted. New structural congruence rules are added to take the restriction of pnames into account. For instance, restriction of pnames and gnames commute (i.e. $(\nu x)(\nu g)N \equiv (\nu g)(\nu x)N$), and the restriction operator can be pushed into or pulled out of node and process expressions as long as free names are not captured. At first glance, it may appear that the structural congruence rules for scope extension of pnames are redundant in the presence of the scope-extrusion rules (OPEN/CLOSE). However, the OPEN/CLOSE rules are essential for reasoning about open systems, and the scope extension rules are essential for defining normal forms; see [16].

The addition of unicast communication raises certain interesting issues with respect to mobility. Recall that *groups* encapsulate the locality of a process. When two processes share a private name, they can use that name as a channel of communication. However, after establishing that link, if the processes move away from each other, they may no longer be able to use that name as a channel. In summary, unicast channels should also respect the locality of communication. We enforce this in the ω_1-calculus by annotating unicast action labels with the interfaces of the participating processes, and allowing synchronization between actions only when their interfaces overlap (meaning that the processes are in each other's transmission range). Hence, the execution of a unicast send action of value x on channel z by a basic node with process interface G is represented by action label $\overline{z:G}x$; the corresponding receive action is labeled $z:G(x)$.

The semantic rules for unicast send (UNI-SEND), receive (UNI-RECV), and synchronization (UNI-COM) are given in Table 4. Scope extrusion via unicast communication is accomplished by naturally extending their π-calculus counterparts (OPEN/CLOSE) rules as follows. Bound-output actions (due to OPEN) are annotated with the interface of the participating process, and the CLOSE rule applies only when the interfaces overlap. These extensions are straightforward, and the details are omitted.

Note that the scope of a name may encompass different processes regardless of their interfaces, and hence two processes may share a secret even when they are outside each others transmission ranges. The restriction we impose is that shared names can be used as unicast channels only when the processes are within each others transmission ranges.

Extending ω_1 to ω_2. We obtain the ω_2-calculus by adding the parallel composition ('|') operator at the process level, thereby allowing concurrent processes within a node. This addition facilitates e.g. the modeling of communication between layers of a protocol stack running at a single node; it also renders the π-calculus a subcalculus of the ω_2-calculus. In ω_2, the actions of two processes within a node may be interleaved. Moreover, two processes within a node can synchronize using unicast (binary) communication. We add PAR, COM and CLOSE rules corresponding to intra-node interleaving, synchronization and scope extrusion, respectively; these rules are straightforward extensions of the corresponding rules in the π-calculus.

4 Bisimulation, Congruence Results and Other Properties of the ω-Calculus

In this section, we prove some fundamental properties of the ω-calculus, including congruence results for strong bisimulation and a weak version of bisimulation that treats τ- and μ-actions as unobservable.

Embedding of the π-Calculus. The ω-calculus is a conservative extension of the π-calculus [11]. That is, every process expression P in the π-calculus can be syntactically translated to an ω-node expression M such that the transition system generated by M directly corresponds to the one generated by P. This property is formally stated by the following theorem, which is readily proved by induction on the length of derivations.

Theorem 1 *Let P be a process expression in the π-calculus. Then $P : \{g\}$ is a node expression in the ω-calculus, where g is a fresh group name not in P. Moreover, $P \xrightarrow{\alpha} P'$ is a transition derivable from the operational semantics of the π-calculus if and only if $P : \{g\} \xrightarrow{\alpha'} P' : \{g\}$ is derivable from the operational semantics of the ω-calculus, and one of the following conditions hold: (i) $\alpha = \alpha' = \tau$; (ii) $\alpha = x(y)$ and $\alpha' = x : \{g\}(y)$; (iii)$\alpha = \overline{x}y$ and $\alpha' = \overline{x : \{g\}}y$; or (iv) $\alpha = (\nu y)\overline{x}y$ and $\alpha' = (\nu y)\overline{x : \{g\}}y$, for some names x, y.*

Decidability of the Finite-Control Fragment. In the *finite-control* fragment of the π-calculus, recursive definitions are not allowed to contain the parallel operator ('|') nor unguarded occurrences of process identifiers. Reachability properties are decidable for closed process expressions (i.e. those without free names) specified in the finite-control fragment [3]. We can extend the notion of finite control to the ω-calculus, and show that reachability remains decidable for closed node expressions. Formally, we say that an ω-calculus expression N is *reachable* from M (denoted by $M \longrightarrow^* N$) if there is a finite sequence of transitions $M \xrightarrow{\alpha_1} M_1 \xrightarrow{\alpha_2} M_2 \cdots \xrightarrow{\alpha_n} N$. We then have the following result.

Theorem 2 *Let M be a finite-control ω-calculus expression such that M is closed w.r.t. names. Then, the set of node expressions reachable from M modulo the structural congruence relation, i.e., $\{N \mid M \longrightarrow^* N\}_{\equiv}$, is finite.*

Theorem 2 is of practical importance in verifying MANET system specifications. Its proof is based on the observation that, in the ω-calculus, the physical notion of

neighborhood is represented abstractly by group-based connectivity information. This ensures that only a finite number of equivalent configurations need be analyzed.

Bisimulation for the ω-calculus. The definition of strong (late) bisimulation for the π-calculus [11] can be extended to the ω-calculus.

Definition 1 *A relation $\mathcal{S} \subseteq \mathbf{N} \times \mathbf{N}$ on nodes is a* strong simulation *if $M\ \mathcal{S}\ N$ implies:*

- *fgn(M) = fgn(N), and*
- *whenever $M \xrightarrow{\alpha} M'$ where $bn(\alpha)$ is fresh then:*
  - *if $\alpha \in \{G(x),\ z{:}G(x)\}$, there exists an N' s.t. $N \xrightarrow{\alpha} N'$ and for each pname y, $M'\{y/x\}\ \mathcal{S}\ N'\{y/x\}$,*
  - *if $\alpha \notin \{G(x),\ z{:}G(x)\}$, there exists an N' s.t. $N \xrightarrow{\alpha} N'$ and $M'\ \mathcal{S}\ N'$.*

$\mathcal{S}$ is a strong bisimulation *if both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are strong simulations. Nodes M and N are* strong bisimilar*, written $M \sim N$, if $M\ \mathcal{S}\ N$, for some* strong bisimulation *$\mathcal{S}$.*

Proposition 3 *(i) $\sim$ is an equivalence; and (ii) $\sim$ is the largest strong bisimulation.*

Strong bisimulation is a congruence for the ω-calculus, as formally stated in Theorem 4.

Theorem 4 (Congruence) *$\sim$ is a congruence relation; i.e., for all nodes $M_1, M_2 \in \mathbf{N}$, the following hold:*

(i) $M_1 \sim M_2$ implies $\forall x \in \mathbf{Pn} : (\nu x)M_1 \sim (\nu x)M_2$;
(ii) $M_1 \sim M_2$ implies $\forall g \in \mathbf{Gn} : (\nu g)M_1 \sim (\nu g)M_2$; and
(iii) $M_1 \sim M_2$ implies $\forall N \in \mathbf{N} : M_1|N \sim M_2|N$.

We have also defined a notion of *weak bisimulation* for the ω-calculus, in which τ- and μ-actions are treated as unobservable. Its definition is similar to that for strong bisimulation (Definition 1) and is given in [16]. There, we also establish that weak bisimulation, like its strong counterpart, is a congruence for the ω-calculus.

5 Case Study: Modeling and Verifying a Leader Election Protocol for MANETs

Syntactic extensions to the ω-calculus. The ω-calculus provides the basic mechanisms needed to model MANETs. In order to make specifications more concise, we extend the calculus to a polyadic version (along the same lines as the polyadic pi-calculus [10]) and also add support for data types such as bounded integers and structured terms. The matching prefix is extended to include equality over these types. Terms composed of these types can be used as values in a unicast or local broadcast transmission, or as actual parameters for a process invocation. The modifications to the theory developed in the preceding sections (Sections 3-4) to account for these syntactic extensions to the calculus are straightforward.

A leader election protocol for MANETs. The algorithm of [17] elects the node with the maximum id among a set of connected nodes as the leader of the connected component. A node that initiates the leader election sends an $election$ message to its neighboring nodes. The recipients of the $election$ message mark the node from which they received the message as their parent and send the $election$ message to their neighbors, thereby building a spanning tree with the initiator as the root. After sending an $election$ message, a node awaits acknowledgements from its children in the spanning tree. A child node n sends its parent an acknowledgement ack with the maximum id in the spanning tree rooted at n. The maximum id in the spanning tree is propagated up the

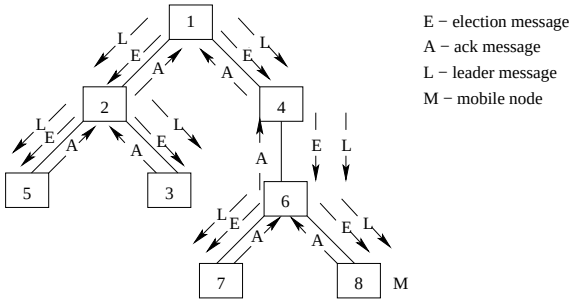

Fig. 3. Message flow in leader election protocol

tree to the root. The root node then announces the leader to all the nodes in its spanning tree by sending a $leader$ message. To keep track of the neighbors of a node, $probe$ and $reply$ messages are used periodically. When a node discovers that it is disconnected from its $leader$, it initiates an election process. The flow of $election$, ack, and $leader$ messages is depicted in Fig. 3, where the node with id 1 is the initiator.

Description of the protocol in the ω-calculus. We model a network as the parallel composition of basic ω-nodes, whose process interfaces reflect the initial topology of the network. Each node runs an instance of process $node(id,\ chan,\ init,\ elec,\ lid,\ pChan)$ defined in Fig. 4. The meaning of this process's parameters is the following: *id* is the node identifier; *chan* is an input channel; *init* indicates whether the node initiates the election process; *elec* indicates whether the node is part of the election process; *lid* represents the node's knowledge of the leader id; and *pChan* is the parent's input channel. These parameters are represented by pnames and integers.

A node may receive $election, ack,$ and $leader$ messages, representing an election message, an acknowledgement to the election process, and a leader message, respectively. We need not consider $probe$ and $reply$ messages in our model because a node can broadcast to its neighbors without knowing its neighbors, and the effect of disconnection between nodes can be modeled using the choice operator. The ω-calculus model of the protocol is given in Fig. 4. The messages, their parameters, and the parameters used in the definitions appearing in Fig. 4 are explained below:

Messages: $election(sndrChan)$; $ack(maxid)$; $leader(maxid)$.

Message parameters: $sndrChan$: input channel of the sender of the message; $maxid$: maximum id seen so far by the sender of the message.

Definition parameters: id: id of the node, $chan$: input channel of the node; $init$: 1 if node initiated the election process, 0 otherwise; $elec$: 1 if node is participating in the election process, 0 otherwise; lid: node's knowledge of the leader id; $pChan$: input channel of the node's parent in the spanning tree; $sndrChan$: input channel of the sender node of the message; $maxid$: maximum id seen so far by the node.

An example specification of an eight-node network running the leader election protocol of Fig. 4 is given in Fig. 5. The initial network topology is the same as that of the network of Fig. 3. The node with id 1 ($initElection$) is designated to be the initiator of the leader-election process. The last parameter $none$ in the process invocations indicates that the parent channel is initially not known to the processes.

/* A node may receive an $election$ or a $leader$ message. */
$node(id, chan, init, elec, lid, pChan) \stackrel{def}{=}$
$\quad \mathbf{r}(election(sndrChan)).\ processElection(id, chan, init, 1, lid, pChan, sndrChan)$
$\quad + \mathbf{r}(leader(maxid)).\ processLeader(id, chan, init, elec, lid, pChan, maxid)$

/* Node that initiates election process broadcasts $election$ msg and awaits ack in state $awaitAck$. */
$initElection(id, chan, init, elec, lid, pChan) \stackrel{def}{=}$
$\quad \overline{\mathbf{b}}\, election(chan).\ awaitAck(id, chan, init, 1, id, none)$

/* When a node receives an $election$ message it reaches the $processElection$ state where it broadcasts the $election$ message and goes to state $awaitAck$. */
$processElection(id, chan, init, elec, lid, pChan, sndrChan) \stackrel{def}{=}$
$\quad \overline{\mathbf{b}}\, election(chan).\ awaitAck(id, chan, init, elec, lid, sndrChan)$

/* A node in $awaitAck$ state may receive an ack and reach $processAck$ state or it may nondeterministically conclude that it has received ack from all its children in the spanning tree. In the latter case, it declares the leader by broadcasting a $leader$ message if it is the initiator. Otherwise, it sends (unicast) an ack to its parent node ($pChan$) with the maximum id in the spanning tree rooted at this node. */
$awaitAck(id, chan, init, elec, lid, pChan) \stackrel{def}{=}$
$\quad chan(ack(maxid)).\ processAck(id, chan, init, elec, lid, pChan, maxid)$
$\quad + [init = 1]\, \overline{\mathbf{b}}\, leader(lid).\ node(id, chan, init, 0, lid, pChan)$
$\quad + [init = 0]\, \overline{pChan}\, ack(id, lid).\ node(id, chan, init, elec, lid, pChan)$

/* On receiving an ack, a node stores the maximum of the ids received in ack messages. */
$processAck(id, chan, init, elec, lid, pChan, maxid) \stackrel{def}{=}$
$\quad [maxid >= lid]\, awaitAck(id, chan, init, elec, maxid, pChan)$
$\quad + [maxid < lid]\, awaitAck(id, chan, init, elec, lid, pChan)$

/* On receiving a $leader$ message, a node sets its lid parameter to the $maxid$ in the $leader$ message. If $maxid$ is less than lid, then either the node was not part of the election process or did not report ack to its parent node (probably because it moved away from its parent). In either case, it broadcasts its lid as the maximum id. */
$processLeader(id, chan, init, elec, lid, pChan, sndrChan, maxid) \stackrel{def}{=}$
$\quad [maxid = lid]($
$\quad\quad [elec = 1]\, \overline{\mathbf{b}}\, leader(maxid).\ node(id, chan, init, 0, lid, pChan)$
$\quad\quad + [elec = 0]\, node(id, chan, init, 0, lid, pChan)$
$\quad)$
$\quad + [maxid > lid]\, \overline{\mathbf{b}}\, leader(maxid).\ node(id, chan, init, 0, maxid, pChan)$
$\quad + [maxid < lid]\, \overline{\mathbf{b}}\, leader(lid).\ node(id, chan, init, 0, lid, pChan)$

Fig. 4. ω-calculus encoding of the leader election protocol for MANETs.

$$
\begin{aligned}
M = {} & (\nu a)(\nu b)(\nu c)(\nu d)(\nu e)(\nu h)(\nu i)(\nu j)(\nu g_1)(\nu g_2)(\nu g_3)(\nu g_4)(\nu g_5)(\nu g_6)(\nu g_7) \\
& (initElection(1, a, 1, 0, 1, none) : \{g_1, g_2\} \\
& \quad | \, node(2, b, 0, 0, 2, none) : \{g_1, g_3, g_4\} \\
& \quad\quad | \, node(3, c, 0, 0, 3, none) : \{g_4\} \\
& \quad\quad\quad | \, node(4, d, 0, 0, 4, none) : \{g_2, g_5\} \\
& \quad\quad\quad\quad | \, node(5, e, 0, 0, 5, none) : \{g_3\} \\
& \quad\quad\quad\quad\quad | \, node(6, h, 0, 0, 6, none) : \{g_5, g_6, g_7\} \\
& \quad\quad\quad\quad\quad\quad | \, node(7, i, 0, 0, 7, none) : \{g_6\} \\
& \quad\quad\quad\quad\quad\quad\quad | \, node(8, j, 0, 0, 8, none) : \{g_7\})
\end{aligned}
$$

Fig. 5. ω-calculus specification of leader election protocol for an 8-node tree-structured network.

Verifying the leader election protocol model. Following our earlier encoding of the semantics of value-passing CCS and the π-calculus [15, 19] using the XSB tabled logic-programming system [18], we encoded the transitional semantics of the ω-calculus using Prolog rules. Each inference rule of the semantics is encoded as a rule for a predicate `trans`, which evaluates the transition relation of a given ω-calculus model. We also encoded a weak bisimulation checker for the ω-calculus in Prolog. The weak version of the transition relation, abstracting τ- and μ-transitions, is encoded as the `dtrans` predicate. The predicate `nb(S1, S2)` checks if two ω-specifications $S1$ and $S2$ are weak bisimilar. Using this implementation, we verified the following correctness property for the leader election protocol for MANETs: *Eventually a node with the maximum id in a connected component is elected as the leader of the component, and every node connected to it (via one or more hops) learns about it.*

The verification was performed on models having *tree*- and *ring*-structured initial topologies. A distinguished node (with maximum id, for example, node 8 marked 'M' for "mobile" in Fig. 3) was free to move as long as the network remained connected. A mobility invariant was used to constrain the other nodes to remain connected to their neighbors. For verification purposes, we added a node *final* to the model that remains connected to all other nodes. A node, upon learning its leader, forwards this information to node *final*. After *final* receives messages from every other node with their leader ids equal to the maximum id in the network, it performs the observable action $action(leader(MaxId))$. The closed ω-specification of the protocol was checked for weak bisimilarity with an ω-specification that emits $action(leader(MaxId))$ as the only observable action. Weak bisimilarity between these two specifications indicates that the correctness property is true of the system.

We verified the correctness property for networks containing 5 through 8 nodes. Table 5 lists the states, transitions and time (in seconds) it took our Prolog implementation of the calculus and weak bisimulation checker to verify the property for networks with initial tree and ring topologies. We consider this implementation to be a prototype. Its main purpose is to demonstrate the feasibility and straightforwardness of implementing the calculus in a tabled logic-programming system. As future work, we plan to develop an optimizing compiler for the ω-calculus, along the lines of one for the π-calculus implemented in the MMC model checker [20]. As these prior results demonstrate, this should significantly improve the performance of our implementation.

Nodes	Tree			Ring		
	States	Transitions	Time(sec)	States	Transitions	Time(sec)
5	77	96	0.97	98	118	1.22
6	168	223	3.35	212	281	4.45
7	300	455	11.55	453	664	17.58
8	663	1073	45.85	952	1560	71.22

Table 5. Verification statistics for ω-calculus model of leader election protocol.

We observed a number of benefits in using the ω-calculus to model the leader election protocol for MANETs. (1) The concise and modular nature of our specification is a direction consequence of the calculus's basic features, including separation of control behavior (processes) from neighborhood information (interfaces), and modeling support for unicast, local broadcast, and mobility. (2) The mobility constraints imposed on the model are specified independently of the control logic using a mobility invariant. For the case at hand, the invariant dictates that all nodes other than a distinguished node (node 8 in Fig. 3) remain connected to their initial neighbors. Thus, during protocol execution, process interfaces may change at will as long as the mobility invariant is maintained. (3) Our specification of the protocol is given in the finite-control sub-calculus of the ω-calculus, thereby rendering it amenable to automatic verification (bisimulation checking); see also Theorem 2.

6 Related Work

Several process calculi have recently been developed for wireless and mobile ad hoc networks. The closest to our work are CBS# [12], CWS [9], CMN [8], and CMAN [5]. These calculi provide local broadcast and separate control behavior from neighborhood information. However, there are significant differences between these calculi and ours, which we now discuss. CBS# [12], based on the CBS process algebra of [14], supports a notion of located processes. Node connectivity information is given independently of a system specification in terms of node connectivity graphs. The effect of mobility is achieved by nondeterministically choosing a node connectivity graph from a family of such graphs when a transition is derived. In contrast, the ω-calculus offers a single, integrated language for specifying control behavior and connectivity information, and permits reasoning about changes to connectivity information within the calculus itself.

In CWS [9], node location and transmission range are a part of the node syntax. Node movement is not supported, although the authors suggest the addition of primitives for this feature. CWS is well-suited for modeling device-level behaviors (e.g., interference due to simultaneous transmissions) in wireless systems.

In CMN [8], a MANET node is a named, located sequential process that can broadcast within a specific transmission radius. Both the location and transmission radius are values in a physical coordinate system. Nodes are designated as mobile or stationary, and those of the former kind can move to an arbitrary location (resulting in a tau-transition). Bisimulation as defined for CMN is based on a notion of physically located observers. A calculus based on physical locations may pose problems for model checking as a model's state space would be infinite if locations are drawn from a real coordinate system.

In CMAN [5], each node is associated with a specific *location*. Furthermore, each node n is annotated by a *connection set*: the set of locations of nodes to which n is

connected. Connections sets thus determine the network topology. Synchronous local broadcast is the sole communication primitive. The connection set of a node explicitly identifies the node's neighbors. Consequently, when a node moves, its neighbors actively participate by removing from (or adding to) their connection sets the location of the moving node. This explicit handling of connection information affects the modularity of the calculus's semantics (the definition of bisimulation, in particular), and may preclude reasoning about open systems. In contrast, in the ω-calculus, neighborhood information is implicitly maintained using groups, thereby permitting us to define bisimulation relations in a natural way.

Other calculi for mobile processes that have been proposed in the literature include the π-calculus [11], $b\pi$-calculus [4], HOBS [13], distributed process calculus $D\pi$ [7], and the ambient calculus [2]. These calculi could be used to model MANETs but not as in a concise and natural fashion as with the ω-calculus.

7 Conclusions and Future Work

The ω-calculus, introduced in this paper, is a conservative extension of the π-calculus that permits succinct and high-level encodings of MANET systems and protocols. The salient aspect of the calculus is its group-based support for local broadcast communication over dynamically changing network topologies. We have shown that reachability of system states is decidable for the finite-control fragment of the calculus, and late bisimulation and its weak counterpart is a congruence. We illustrated the practical utility of the new formalism by using it to develop a model of a leader-election algorithm for MANETS [17]. We also showed how the calculus's operational semantics can be readily encoded in the XSB tabled logic-programming system, thereby allowing us to generate transition systems from ω-calculus specifications. We used this feature to implement a weak bisimulation checker for the ω-calculus, which we then used to verify certain key properties of our encoding of the leader election algorithm of [17].

We have also considered the problem of adding a π-calculus-like *mismatch* operator to the ω-calculus [16], the introduction of which necessitates a lifting of the calculus's transitional semantics to a symbolic one. This is to ensure that terms identified as unequal do not violate substitution of free names in expressions. As desired, the congruence results of Section 4 can be established for this extension as well [16].

As mentioned in Section 5, future work involves the development of an optimizing compiler for the ω-calculus, along the lines of one for the π-calculus implemented in the MMC model checker [20]. MMC exploits the use of binary synchronization in the π-calculus, generating specialized rules from which the transition system can be derived efficiently at model-checking time. The MMC compiler enables MMC to match the efficiency of model checkers for non-mobile systems. Extending such compilation techniques to broadcast and multicast communication is an open problem. Another avenue of future work is the development of a compositional model checker for the ω-calculus, such as of those for CCS and the π-calculus [1, 21]. A model checker of this nature would permit verification of infinite families of MANETs. Finally, the ω-calculus models bidirectional connectivity between nodes. Since certain MANET protocols rely on unidirectional node connections, it would be fruitful to extend the calculus with such a modeling capability.

Acknowledgements. We thank Massimo Merro for his valuable comments on an earlier version of this paper. Research supported in part by NSF grants CCR-0205376, CNS-0509230, CNS-0627447, and ONR grant N000140710928.

Bibliography

[1] S. Basu and C. R. Ramakrishnan. Compositional analysis for verification of parameterized systems. In *TACAS*, volume 2619 of *LNCS*, pages 315–330. Springer, 2003.

[2] L. Cardelli and A. D. Gordon. Mobile ambients. In *FOSSACS*. Springer-Verlag, 1998. URL `citeseer.ist.psu.edu/cardelli98mobile.html`.

[3] M. Dam. On the decidability of process equivalences for the π-calculus. *Theoretical Computer Science*, 183:215–228, 1997. URL `http://www.sics.se/~mfd/tcs97bisim.ps`.

[4] C. Ene and T. Muntean. A broadcast-based calculus for communicating systems. In *Intl. Workshop on Formal Methods for Parallel Programming: Theory and Applications*, 2001. URL `citeseer.csail.mit.edu/ene00broadcastbased.html`.

[5] J. C. Godskesen. A calculus for mobile ad hoc networks. In *COORDINATION*, volume 4467 of *Lecture Notes in Computer Science*, pages 132–150. Springer, 2007.

[6] J. C. Godskesen and O. Gryn. Modelling and verification of security protocols for ad hoc networks using UPPAAL. In *Proc. 18th Nordic Workshop on Programming Theory*, Oct. 2006.

[7] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. In *High-Level Concurrent Languages*, volume 16.3 of *Electr. Notes Theor. Comput. Sci.*, pages 3–17, 1998. URL `citeseer.ist.psu.edu/hennessy98resource.html`.

[8] M. Merro. An observational theory for mobile ad hoc networks. In *Proc. MFPS '07*, volume 173 of *Electr. Notes Theor. Comput. Sci.*, pages 275–293. Elsevier, 2007.

[9] N. Mezzetti and D. Sangiorgi. Towards a calculus for wireless systems. In *Proc. MFPS '06*, volume 158 of *Electr. Notes Theor. Comput. Sci.*, pages 331–354. Elsevier, 2006.

[10] R. Milner. The polyadic pi-calculus: a tutorial. In *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993. URL `citeseer.ist.psu.edu/milner91polyadic.html`.

[11] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Parts I and II. *Information and Computation*, 100(1):1–77, 1992.

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

[13] K. Ostrovsky, K. V. S. Prasad, and W. Taha. Towards a primitive higher order calculus of broadcasting systems. In *PPDP*, pages 2–13. ACM, 2002. ISBN 1-58113-528-9. doi: http://doi.acm.org/10.1145/571157.571159.

[14] K. V. S. Prasad. A calculus of broadcasting systems. *Sci. Comput. Program.*, 25 (2-3):285–327, 1995.

[15] Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. Swift, and D. S. Warren. Efficient model checking using tabled resolution. In *CAV*, volume 1254 of *LNCS*, pages 143–154. Springer, 1997.

[16] A. Singh, C. R. Ramakrishnan, and S. A. Smolka. A process calculus for mobile ad hoc networks, 2008. `http://www.lmc.cs.sunysb.edu/~anusingh/omega/`.

[17] S. Vasudevan, J. F. Kurose, and D. F. Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. In *ICNP*, pages 350–360. IEEE Computer Society, 2004. ISBN 0-7695-2161-4.

[18] XSB. The XSB logic programming system. `http://xsb.sourceforge.net`.

[19] P. Yang, C. R. Ramakrishnan, and S. A. Smolka. A logical encoding of the pi-calculus: Model checking mobile processes using tabled resolution. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(1):38–66, 2004.

[20] P. Yang, Y. Dong, C. R. Ramakrishnan, and S. A. Smolka. A provably correct compiler for efficient model checking of mobile processes. In *PADL*, volume 3350 of *LNCS*, pages 113–127. Springer, 2005.

[21] P. Yang, S. Basu, and C. R. Ramakrishnan. Parameterized verification of pi-calculus systems. In *TACAS*, volume 3920 of *LNCS*, pages 42–57. Springer, 2006.