Available online at www.sciencedirect.com

ScienceDirect

Theoretical Computer Science 367 (2006) 203–227

Theoretical Computer Science

www.elsevier.com/locate/tcs

A framework for security analysis of mobile wireless networks

Sebastian Nanz*,[1], Chris Hankin

Department of Computing, Imperial College London, UK

Abstract

We present a framework for specification and security analysis of communication protocols for mobile wireless networks. This setting introduces new challenges which are not being addressed by classical protocol analysis techniques. The main complication stems from the fact that the actions of intermediate nodes and their connectivity can no longer be abstracted into a single unstructured adversarial environment as they form an inherent part of the system's security. In order to model this scenario faithfully, we present a broadcast calculus which makes a clear distinction between the protocol processes and the network's connectivity graph, which may change independently from protocol actions. We identify a property characterising an important aspect of security in this setting and express it using behavioural equivalences of the calculus. We complement this approach with a control flow analysis which enables us to automatically check this property on a given network and attacker specification.

Keywords: Process calculi; Control flow analysis; Security analysis; Mobile ad hoc networks

1. Introduction

In classical cellular wireless networking, devices connect to a dedicated base station providing services such as Internet access. Considering the threats implied in using the wireless medium, security has always been a natural concern in this setting. In order to increase convenience and mobility of users even further, much research effort has been spent in recent years on the development of protocols for networks operating without central control components so that nodes connect directly to each other and forward messages over multiple hops. This networking paradigm is dubbed "mobile ad hoc networking" and some of the proposed protocols have been standardised by the Internet Engineering Task Force (IETF) or are in the process of standardisation.

Security is again a major concern in this new setting, however, the added complexity asks for new security models and properties which do not yet seem to be well understood. For example, the signature mechanism of the "secure" routing protocol extension SAODV [13] clearly strives for the authentication of endpoint nodes of a yet to be established routing path. However, the task of any routing protocol is to discover and maintain paths between communication partners in a network, and this service itself should be secured. As SAODV's authentication property does not imply statements about paths, it is unclear how the securing of the service is to be achieved.

To ensure the correctness of any protocol, formal modelling and analysis techniques are to be employed. This approach has proved to be successful with security protocols for properties like authentication and confidentiality,

* Corresponding author. Tel.: +44 20 759 48291; fax: +44 20 758 1802.
E-mail addresses: nanz@doc.ic.ac.uk (S. Nanz), clh@doc.ic.ac.uk (C. Hankin).

[1] Supported in part by a doctoral fellowship of the DAAD.

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.08.036

and a multitude of effective frameworks have been proposed, e.g. [8,24,2,12] to name only a few. The above example reveals, however, that these formalisms cannot be applied in this new setting because they are designed for endpoint properties. The goal of this paper is to present a framework, based on a broadcast calculus and static analysis, which allows mobile wireless networks and their security to be formally described and analysed. In the following, we review some of the characteristics of mobile ad hoc networking, summarise related work and our contributions, and give an overview of the structure of this paper.

1.1. Background: mobile ad hoc networks

Mobile ad hoc networks consist of mobile devices communicating via wireless transmission. Nodes cooperate by relaying messages to distant partners, thus eliminating the need for any pre-installed infrastructure and overcoming the limitations of their respective radio transmission ranges. In order to achieve this behaviour, multiple protocols have to work together. Their design is carried out within a layered architecture, so that protocols on higher layers can abstract from functionality supplied by those beneath. For mobile ad hoc networks, the network layer is critical as routing is the central issue.

Routing comprises two complementary tasks: route discovery and route maintenance. For *route discovery*, a node usually floods the network with a route request message which is rebroadcast over and over by intermediate nodes until the destination node is found and can acknowledge. Found routes are kept in routing tables for later use. *Route maintenance* tries on the other hand to repair routes (by finding alternatives) whenever a link between nodes on a route breaks. Broken links are considered to occur frequently, as nodes are free to move about.

Routing protocols are further distinguished into *proactive* (nodes constantly try to update their routing tables according to the changing network topology) and *on-demand* (routes are on), with the proactive approach seen as less advantageous because it produces a greater routing overhead on the network. Protocols under standardisation at the IETF include ad hoc on-demand distance vector (AODV) routing [25] and dynamic source routing (DSR) [16], both on-demand protocols. These are developed for a non-adversarial setting only, and security extensions such as SAODV [14], Ariadne [15], and ARAN [29] have been proposed to secure the routing effort.

To summarise, main characteristics of mobile ad hoc networks include the following and clarify the issues any modelling formalism for this setting has to address:

Internal states: Nodes are not memoryless but store information in routing tables with impact on future actions.

Broadcast communication: Routing protocols rely on broadcast as the main mode of communication, together with some notion of locality: only adjacent nodes receive the initial broadcast message.

Connectivity: Connectivity of nodes is a separate parameter to the system and protocols promise to give correct results for any value of this parameter.

Dynamic environment: The connectivity undergoes constant changes as the result of link failures.

1.2. Related work

There are not many works on bringing together the development of protocols for mobile ad hoc networks with formal modelling and security analysis (our own preliminary studies are [18,19]). Related work is thus to be found mainly in the separate areas of process algebra, protocol analysis, and non-formal security analysis for mobile ad hoc networks.

In the realm of process algebra, broadcast calculi, distributed calculi, and calculi with security objectives are most closely related to our work. The *calculus of broadcasting systems* (*CBS*) [27] is the first calculus to have broadcast as communication primitive, and is a direct ancestor of our calculus. As a main difference to our approach, all processes receive a broadcast message at once, whereas we emphasise the necessity of a notion of local broadcast in which only adjacent nodes can directly receive a transmitted message. The *bπ-calculus* [11] equips the π-calculus with a broadcast paradigm such that only nodes listening on the right channel will receive a broadcast. While this seems to come closer to a notion of local broadcast, it remains complicated to change a once established connectivity (which is straightforward in our calculus).

Process calculi have been enriched with locations to describe distributed mobile computation. *Mobile Ambients* [9] model located places for parallel computation which can be nested in order to reason about process mobility. *Klaim* [3] is a language which owes concepts to both process algebra and coordination languages. Locality variables and a so-called allocation environment permit writing programs for distributed environments while ignoring their precise allocations. While we use locations and a style of notation similar to mobile ambients, our approach distinguishes itself from these and similar calculi by relating locations and connectivity specifically for the purpose of modelling wireless communication.

In the area of calculi with security objectives, the *spi calculus* [2] was the first calculus to include explicit cryptographic primitives. Our approach follows however more closely the *applied pi calculus* [1] which allows functions as term constructors and uses them to model cryptography. In both approaches security is expressed with the help of behavioural equivalences for processes. The calculi *LySa* [5] and $LySa^{NS}$ [7] are spi calculus variants with powerful modelling constructs (pattern matching) which focus on establishing security results directly via static analysis without developing behavioural equivalences.

Model checking has been used to analyse traditional security protocols specified in a process algebra framework [28]. Static analysis techniques for the same task have been employed by Bodei et al. [5] and stimulated our own approach. Bhargavan et al. [4], Zakiuddin et al. [32], Chiyangwa et al. [10] and Wibling et al. [31] have used model checking to discover flaws in routing protocols for mobile ad hoc networks, but consider only safety problems.

Work on the security of mobile ad hoc networks includes [14,15,29], which propose secure protocols or protocol extensions and informally consider attacks and desired properties.

1.3. Contributions

Our main contributions can be summarised as follows:

- Definition of the calculus $\mathrm{CBS}^\sharp$ for the faithful formalisation of protocols for mobile wireless networks. Its main novelties include: *local broadcast* as main communication model; separation of process connectivity (represented by graphs on locations) and process actions; explicit notion of a computational entity as a pair of process and private store at some location.
- The definition of *topology consistency* as a building block for routing security, formalised using the notion of *mediated process equivalence* which focuses on identifying processes only with respect to their capabilities to store items.
- A static analysis to overapproximate actions of a finite network of nodes specified in $\mathrm{CBS}^\sharp$, which can be used to automatically derive the topology consistency condition.

1.4. Outline of the paper

In Section 2, the syntax and operational semantics of the calculus $\mathrm{CBS}^\sharp$ are defined and we present related behavioural equivalences. Section 3 presents a control flow analysis on terms of our calculus to yield an overapproximation of the sets of terms transmitted and stored in a network. We prove the analysis correct with respect to the operational semantics of the calculus and give notes about our implementation of the analysis. In Section 4, we analyse the security needs of mobile wireless networks and compare them with classical security protocols. We specify μSAODV, a simplified version of SAODV, as a worked example of protocol modelling in $\mathrm{CBS}^\sharp$. We formalise topology consistency as an important property in the setting of routing protocols and apply our analysis to show that μSAODV violates this property. We conclude with notes on future work in Section 5.

2. The calculus $\mathrm{CBS}^\sharp$

In this section we present $\mathrm{CBS}^\sharp$, a process calculus for modelling mobile wireless networks. It inherits broadcast as the base communication paradigm from the CBS [27], with the important difference that sent messages are not received globally, but only by adjacent neighbours of the sending node. The notion of adjacency is made explicit by the concept of a connectivity graph, which effectively separates process actions from process connectivity. This separation is essential for modelling mobile wireless networks since the possibility of a connection is determined by environment conditions such as node movement, but never by process actions.

For security modelling, we develop notions of behavioural equivalences of networks, much in the style of the spi calculus [2]. However, security will be determined by the tense relation between the actual environment conditions and what nodes believe about their environment (see later Section 4.3). The belief of the nodes can be expressed in

Table 1
Syntax of CBS$^\sharp$

Terms	
$T ::= n$	Names, $n \in \mathcal{N}$
$\quad \mid\ x$	Variables, $x \in \mathcal{X}$
$\quad \mid\ f(\widetilde{T})$	Function application, $f \in \mathcal{F}$
Processes	
$P ::= \textbf{nil}$	Termination
$\quad \mid\ \textbf{out}\ T.\,P$	Sending
$\quad \mid\ \textbf{in}\ x.\,P$	Reception
$\quad \mid\ \textbf{store}\ T.\,P$	Storage
$\quad \mid\ \textbf{read}\ x.\,P$	Retrieval
$\quad \mid\ \textbf{case}\ T\ \textbf{of}\ f(\widetilde{T};\widetilde{x})\ P_1\ \textbf{else}\ P_2$	Case distinction, $f \in \mathcal{F}$
$\quad \mid\ P_1 \mid P_2$	Parallel composition
$\quad \mid\ !P$	Replication
Networks	
$N ::= n[P, S]$	Node, $n \in \mathcal{N}_{loc} \subseteq \mathcal{N}$, S non-empty set of terms
$\quad \mid\ N_1 \parallel N_2$	Parallel composition

CBS$^\sharp$ by modelling routing tables using the notion of a private store, and the notion of mediated equivalence focuses on identifying networks by their storage capabilities.

2.1. Syntax and informal semantics

The syntax of CBS$^\sharp$ is given in Table 1.

Terms: Let $\mathcal{N}$ denote a countable set of names, $\mathcal{X}$ a countable set of variables, and $\mathcal{F}$ a finite set of function symbols together with a function arity : $\mathcal{F} \rightarrow \mathbb{N}$ to yield the arity of a function symbol. The set of terms $\mathbf{T}$ consists of names $n \in \mathcal{N}$, variables $x \in \mathcal{X}$, and function applications $f(T_1, \ldots, T_k)$ where $T_i \in \mathbf{T}$, $f \in \mathcal{F}$ and $\text{arity}(f) = k$.

We write $\widetilde{T}$ to abbreviate a finite sequence of terms $T_1, \ldots, T_k$ for some $k \in \mathbb{N}$, and use the function $|\,.\,|$ to denote its arity, $|\widetilde{T}| = k$. Function *fv* yields the set of free variables of a term, process, or node. A free variable x can be replaced with a term U by the substitution $[U/x]$. We write $[\widetilde{U}/\widetilde{x}]$ and $[U_i/x_i]_{i=1}^{k}$ to denote the sequence of substitutions $[U_1/x_1]\ldots[U_k/x_k]$.

Processes: The set of processes $\mathbf{P}$ is inductively defined as follows: the terminated process is represented by **nil**. The sending of a ground term T is denoted by **out** $T.\,P$, with P as the continuation process. The sending mode is *local broadcast*, i.e. only adjacent nodes may receive the transmission, where adjacency is defined below via the notion of connectivity graphs. The action **in** $x.\,P$ expresses readiness to receive a ground term T and then to continue as P with T substituted for x.

The action **store** $T.\,P$ denotes storage of a ground term T in a private store S introduced below. Terms are retrieved from the store by the action **read** $x.\,P$ which non-deterministically chooses a term $T \in S$ then continues as P with T substituted for x. A form of matching can be used to select terms as demonstrated in Section 2.3.

The action for case distinction **case** T **of** $f(\widetilde{T};\widetilde{x})$ P_1 **else** P_2 tries to match a term T with the term $f(\widetilde{T};\widetilde{x})$ and continues on success with P_1 where $\widetilde{U}$ is substituted for $\widetilde{x}$, or otherwise with P_2. In order to match, T has to be of the form $f(\widetilde{T}, \widetilde{U})$ with $|\widetilde{U}| = |\widetilde{x}|$. Processes can be executed in parallel, written as $P_1 \mid P_2$. Multiple parallel actions are abbreviated by $|_{i \in \mathcal{I}}\, P_i$. $!P$ is the operator which allows to express infinite behaviours by producing as many copies of P as are needed.

Stores: Stores are non-empty sets of terms. The set constructor $\{.\}$ and the operations union $\cup$ and element $\in$ are defined for them and enjoy the usual properties. For singleton sets, the set brackets can be dropped. For instance, the union of singleton sets T_1 and T_2 can be written $T_1 \cup T_2$ or $\{T_1, T_2\}$. Stores are defined non-empty to prevent the read operation from getting stuck. If no particular initialisation of stores is wished for, this can be achieved by singleton stores ε, where ε is a distinguished term, called the *empty term*. Unless otherwise specified, we will assume this initialisation.

Networks: Networks $N \in \mathbf{N}$ consist of nodes which are written as $n[P, S]$ and denote a computational entity of a network: a pair of a process P and a private store S at some location n. Locations are contained in the set $\mathcal{N}_{loc} \subseteq \mathcal{N}$ and make it possible to identify nodes. For a network N we define $V(N)$ to yield the set of all locations (vertices) in N. Networks can be composed in parallel by $N_1 \parallel N_2$, where again multiple composition can be written $\|_{i \in \mathcal{I}} N_i$.

2.2. *Operational semantics*

2.2.1. *Connectivity graphs and network topologies*

A graph is a pair $G = (V, E)$ where V is a set, called the set of vertices, and E is a set of unordered pairs of vertices, called the set of edges, with $(m, n) \in E$ implies $m \neq n$ (no self-loops). The set of vertices of a graph G can be referred to as $V(G)$, and the set of edges as $E(G)$.

A *connectivity graph* G is a graph whose vertex set is a subset of $\mathcal{N}_{loc}$. Connectivity graphs are used to describe the connections between nodes for a particular moment in time. G is said to be *admissible* on a network N iff $V(N) \subseteq V(G)$.

A *network topology* $\mathcal{T}$ is a non-empty collection of connectivity graphs. The network topology is used to implicitly describe connectivity properties which remain invariant over time, for example, if a certain link always or never exists. $\mathcal{T}$ is said to be admissible on a network N iff all $G \in \mathcal{T}$are. In the following we will assume that all examined connectivity graphs and network topologies are admissible on their respective networks. The *maximal network topology* $\mathcal{T}_{\max}$ contains all graphs on $\mathcal{N}_{loc}$.

2.2.2. *Transition relations*

The operational semantics of the calculus is defined by the following transition relations:

$N \xrightarrow{(U,m)\sharp}_G N'$	Labelled transition relation
$N \to N'$	Reduction relation
$N \equiv N'$	Structural equivalence

Labelled transition relation: The *labelled transition relation* $N \xrightarrow{(U,m)\sharp}_G N'$ describes the evolution of a network N to a network N' during sending of the ground term U by node m, where N *abides by* a connectivity graph G: only nodes n with $(m, n) \in E(G)$ may receive U. The explanation of the labelling requires the following definition:

Definition 2.1 (*Mode identifiers*). The set {!, ?, :} is called the set of *mode identifiers*. They can be composed with the operator $\circ$ according to the algebra shown below, where $\perp$ means that ! cannot be combined with itself.

$\circ$	!	?	:
!	$\perp$	!	!
?	!	?	?
:	!	?	:

Mode identifiers are ranged over by the meta variable $\sharp$.

In a label $(U, m)\sharp$, the following communication modes can be expressed: sending $(U, m)!$, reception $(U, m)?$, and loss (U, m): of the term U sent out by m.

The relation $\xrightarrow{(U,m)!}_G$ describes network evolution under the sending of one message (U, m) under G. The sending of k messages is serialised in the sense that k derivations for a sequence of connectivity graphs $G_1, \ldots, G_k$ have to be found:

$$N \xrightarrow{(U_1,m_1)!}_{G_1} N_1 \xrightarrow{(U_2,m_2)!}_{G_2} N_2 \cdots \xrightarrow{(U_k,m_k)!}_{G_k} N'$$

Note that this means that there will never be clashes of messages, but sending remains globally asynchronous.

Table 2
Labelled transition relation

Nil	$n[\mathbf{nil}, S] \xrightarrow{(U,m):}_G n[\mathbf{nil}, S]$
Out$_1$	$n[\mathbf{out}\ T.\,P, S] \xrightarrow{(T,n)!}_G n[P, S]$
Out$_2$	$n[\mathbf{out}\ T.\,P, S] \xrightarrow{(U,m):}_G n[\mathbf{out}\ T.\,P, S]$
In$_1$	$\dfrac{(m,n) \in E(G)}{n[\mathbf{in}\ x.\,P, S] \xrightarrow{(U,m)?}_G n[P[U/x], S]}$
In$_2$	$\dfrac{(m,n) \notin E(G)}{n[\mathbf{in}\ x.\,P, S] \xrightarrow{(U,m):}_G n[\mathbf{in}\ x.\,P, S]}$
Repl$_1$	$n[\mathbf{!}P, S] \xrightarrow{(U,m):}_G n[\mathbf{!}P, S]$
PPar	$\dfrac{N_1 \xrightarrow{(U,m)\sharp_1}_G N_1' \qquad N_2 \xrightarrow{(U,m)\sharp_2}_G N_2'}{N_1 \parallel N_2 \xrightarrow{(U,m)(\sharp_1 \circ \sharp_2)}_G N_1' \parallel N_2'}$ where $\circ$ is due to Definition 2.1
Struct	$\dfrac{N \equiv M \qquad M \xrightarrow{(U,m)\sharp}_G M' \qquad M' \equiv N'}{N \xrightarrow{(U,m)\sharp}_G N'}$

In order to ensure that changes in connectivity are in agreement with the network topology, we define the following:

Definition 2.2 (*$\mathcal{T}$-Faithfulness*). The relation $\xrightarrow{(U,m)\sharp}_G$ is called *$\mathcal{T}$-faithful* iff $G \in \mathcal{T}$, and we write $\xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}}$ to emphasise this fact.

Moreover, a finite, possibly empty sequence of $\mathcal{T}$-faithful sending transitions

$$N \xrightarrow{(U_1,m_1)!}_{G_1\in\mathcal{T}} N_1 \xrightarrow{(U_2,m_2)!}_{G_2\in\mathcal{T}} N_2 \cdots \xrightarrow{(U_k,m_k)!}_{G_k\in\mathcal{T}} N'$$

can be written as $N \longrightarrow^*_{\mathcal{T}} N'$.

After this overview, we can now proceed to the rules for the labelled transition relation which are given in Table 2.

Rule Nil expresses that the node running the nil process $n[\mathbf{nil}, S]$ loses any message (U, m). Rules Out$_1$ and Out$_2$ describe the behaviour of a sender $n[\mathbf{out}\ T.\,P, S]$. Such a node loses incoming messages (U, m) and evolves to $n[P, S]$ when broadcasting its own message. In the latter case, the transition arrow is labelled with $(T, n)!$, showing the transmitted term T and the location of the sender n.

There are two rules for a receiver $n[\mathbf{in}\ x.\,P, S]$ which are distinguished by the properties of the connectivity graph G. According to rule In$_1$, the message (U, m) is received if there is an edge in G between location m, the sender of the message, and the current node's location n. In this case the variable x in P is replaced by U so that the continuation process is $P[U/x]$. If, however, $(m, n) \notin E(G)$, any message will be lost by rule In$_2$.

A node running the replication process $n[\mathbf{!}P, S]$ evolves to itself and loses any message (U, m), as described by rule Repl$_1$. Note that rule Repl$_2$ (see Table 3, reduction relation) provides for the expected expansion $\mathbf{!}P = P \mid \mathbf{!}P$.

The rule for parallelism PPar makes use of the algebra for the composition $\circ$ of $\sharp_1, \sharp_2 \in \{!, ?, :\}$ which is due to Definition 2.1. This is essential for the working of the broadcast modelling, as it makes sure that every subprocess running at every node in the network will decide about receiving or losing a particular message.

It is easy to see that the properties "$! \circ ! = \bot$" (i.e., $! \circ !$ is not allowed) and "$\sharp_1 \circ \sharp_2 \in \{?, :\} \Rightarrow \sharp_1, \sharp_2 \neq !$" of the algebra in Definition 2.1 ensure that at most one process can execute a sending action in any transition step.

Table 3
Reduction relation

Repl$_2$	$n[!P, S] \;\rightarrow\; n[P \mid !P, S]$
Store	$n[\textbf{store}\ T.\, P, S] \;\rightarrow\; n[P, S \cup T]$
Read	$\dfrac{T \in S}{n[\textbf{read}\ x.\, P, S] \;\rightarrow\; n[P[T/x], S]}$
Case$_1$	$\dfrac{T = f(\widetilde{T}, \widetilde{U}) \quad \lvert\widetilde{U}\rvert = \lvert\widetilde{x}\rvert}{n[\textbf{case}\ T\ \textbf{of}\ f(\widetilde{T}; \widetilde{x})\ P_1\ \textbf{else}\ P_2, S] \;\rightarrow\; n[P_1[\widetilde{U}/\widetilde{x}], S]}$
Case$_2$	$\dfrac{\nexists \widetilde{U}.\, T = f(\widetilde{T}, \widetilde{U}) \wedge \lvert\widetilde{U}\rvert = \lvert\widetilde{x}\rvert}{n[\textbf{case}\ T\ \textbf{of}\ f(\widetilde{T}; \widetilde{x})\ P_1\ \textbf{else}\ P_2, S] \;\rightarrow\; n[P_2, S]}$

Example 2.3. Let three nodes be defined as follows:

$$N_1 \overset{\text{def}}{=} n_1[\textbf{out}\ T.\,\textbf{nil}, S_1]\ ,\ N_2 \overset{\text{def}}{=} n_2[\textbf{in}\ x.\,\textbf{nil}, S_2]\ ,\ N_3 \overset{\text{def}}{=} n_3[\textbf{in}\ x.\,\textbf{nil}, S_3]\ ,$$

and let $E(G_1) = \{(n_1, n_2)\}$ and $\mathcal{T} = \{G_1\}$. Then the network consisting of the parallel composition of the nodes can evolve according to the following derivation:

$$\dfrac{N_1 \xrightarrow{(T,n_1)!}_{G_1 \in \mathcal{T}} n_1[\textbf{nil}, S_1] \quad \dfrac{N_2 \xrightarrow{(T,n_1)?}_{G_1 \in \mathcal{T}} n_2[\textbf{nil}, S_2] \qquad N_3 \xrightarrow{(T,n_1):}_{G_1 \in \mathcal{T}} N_3}{N_2 \parallel N_3 \xrightarrow{(T,n_1)?}_{G_1 \in \mathcal{T}} n_2[\textbf{nil}, S_2] \parallel N_3}}{N_1 \parallel N_2 \parallel N_3 \xrightarrow{(T,n_1)!}_{G_1 \in \mathcal{T}} n_1[\textbf{nil}, S_1] \parallel n_2[\textbf{nil}, S_2] \parallel N_3}$$

If on the other hand $\mathcal{T} = \{G_1, G_2\}$, where $E(G_2) = \{(n_1, n_3)\}$, then we can conclude that the final configuration will either be

$$n_1[\textbf{nil}, S_1] \parallel n_2[\textbf{in}\ x.\,\textbf{nil}, S_2] \parallel n_3[\textbf{nil}, S_3]$$

or

$$n_1[\textbf{nil}, S_1] \parallel n_2[\textbf{nil}, S_2] \parallel n_3[\textbf{in}\ x.\,\textbf{nil}, S_3]\ .$$

Note that $\mathcal{T}$ models in this case the particular scenario in which either n_2 or n_3 are able to receive T. If we were to model full mobility instead (so that also both or neither of the n_2 and n_3 might receive T), additional network connectivity graphs would have to be added to $\mathcal{T}$.

Reduction relation: We display rules for the reduction relation in Table 3.

$n[!P, S]$ reduces to $n[P \mid !P, S]$, thereby producing one copy of process P. The interplay of Red,Par, and Trans of the structural equivalence makes sure that as many copies of P can be produced as needed. In rule Store, $n[\textbf{store}\ T.\, P, S]$ adds a term T to the store S. Rule Read for retrieval $n[\textbf{read}\ x.\, P, S]$ replaces x in P by an arbitrary term $T \in S$. In the rules for case distinction, the term T is matched against a template $f(\widetilde{T}; \widetilde{x})$. In rule Case$_1$, if T is of the form $f(\widetilde{T}, \widetilde{U})$ and $\lvert\widetilde{U}\rvert = \lvert\widetilde{x}\rvert$, the continuation binds the terms $\widetilde{U}$ against the variables $\widetilde{x}$ to yield $n[P_1[\widetilde{U}/\widetilde{x}], S]$. Otherwise, rule Case$_2$ demands P_2 as continuation process.

Example 2.4. Assume the following definition:

$$N \overset{\text{def}}{=} n[\textbf{case}\ T\ \textbf{of}\ \mathsf{Env}(n; x)\ \textbf{store}\ x.\,\textbf{nil}\ \textbf{else}\ \textbf{nil}, S]\ .$$

Table 4
Structural equivalence

Comm	$N_1 \parallel N_2 \equiv N_2 \parallel N_1$
Assoc	$N_1 \parallel (N_2 \parallel N_3) \equiv (N_1 \parallel N_2) \parallel N_3$
Refl	$N \equiv N$
Sym	$\dfrac{N' \equiv N}{N \equiv N'}$
Trans	$\dfrac{N \equiv N'' \quad N'' \equiv N'}{N \equiv N'}$
Comp	$\dfrac{N_1 \equiv N_1'}{N_1 \parallel N_2 \equiv N_1' \parallel N_2}$
Red	$\dfrac{N \ \rightarrow\ N'}{N \equiv N'}$
Par	$n[P_1 \mid P_2, S_1 \cup S_2] \ \equiv n[P_1, S_1] \ \parallel n[P_2, S_2]$

For $T = \mathsf{Env}(n; msg)$, an envelope addressed to n with contents msg, we have the following derivation:

$$N \ \rightarrow\ n[\textbf{store}\ msg.\,\textbf{nil}, S] \ \rightarrow\ n[\textbf{nil}, S \cup msg]\ .$$

For $T = \mathsf{Env}(m; msg)$ we would however get $n[\textbf{nil}, S]$ as final configuration, since $n \neq m$.

Structural equivalence: The rules of the structural equivalence on nodes are shown in Table 4.

Rules Comm through Trans are standard. We regard networks as structurally equivalent, if one of them can be reduced to the other, as shown in rule Red. Rule Par says that two parallel processes running at location n can be viewed as two separate nodes at n which run these processes and have access to the contents of the original store as necessary.

Example 2.5. We have the following equivalences, where the second holds because of Comp, Red, and Store, the other two because of Par.

$$\begin{aligned} n[\textbf{store}\ T.\,\textbf{nil} \mid \textbf{in}\ x.\,\textbf{nil}, S] \ &\equiv n[\textbf{store}\ T.\,\textbf{nil}, S] \ \parallel n[\textbf{in}\ x.\,\textbf{nil}, S] \\ &\equiv n[\textbf{nil}, S \cup T] \ \parallel n[\textbf{in}\ x.\,\textbf{nil}, S] \ \equiv n[\textbf{nil} \mid \textbf{in}\ x.\,\textbf{nil}, S \cup T]\ . \end{aligned}$$

2.3. Notational conventions and cryptographic primitives

For the specification of even moderately large protocols such as μSAODV in Section 4.2, clarity and readability of the formalisation are imperative. For this reason, we have opted for keywords such as **in** and **read** rather than just symbols which are favoured by other calculi. We also use intelligible identifiers for names, variables, and functions, and omit a trailing **nil** whenever no confusion arises. In this section we introduce some more notational conventions and show a way to model cryptographic primitives in CBS$^\sharp$.

Notation: If the equality of terms is to be checked, the full power of the matching mechanism of the case statement is not needed and a simpler representation is desirable. This can be done with the following encoding which uses the special function Match to allow arbitrary terms T and U, since case can only match terms which a function has been applied to.

$$\textbf{if}\ T = U\ \textbf{then}\ P_1\ \textbf{else}\ P_2 \equiv \textbf{case}\ \mathsf{Match}(T)\ \textbf{of}\ \mathsf{Match}(U;\,)\ P_1\ \textbf{else}\ P_2$$

If **in** and **read** are directly followed by a case distinction on their input variable, we use the following simplified notation.

$$\begin{array}{lll} \textbf{in}\ f(\widetilde{T};\widetilde{x}).\,P & \equiv \textbf{in}\ x.\,\textbf{case}\ x\ \textbf{of}\ f(\widetilde{T};\widetilde{x})\ P\ \textbf{else nil}, & x \notin fv(P) \\ \textbf{read}\ f(\widetilde{T};\widetilde{x})\ P_1\ \textbf{else}\ P_2 & \equiv \textbf{read}\ x.\,\textbf{case}\ x\ \textbf{of}\ f(\widetilde{T};\widetilde{x})\ P_1\ \textbf{else}\ P_2, & x \notin fv(P) \end{array}$$

Cryptographic primitives: In order to express public-key digital signatures we use key pairs (PubKey(*seed*), PrivKey(*seed*)) created from the same *seed* by applying functions PubKey and PrivKey. A signature of term T under private key PrivKey(*seed*) then simply corresponds to applying the function Sign to yield Sign(PrivKey(*seed*), T). Checking of the signature amounts to verifying that the seeds of the known public key and the private key used for the encryptions are the same, and that the right term T has been signed. As shown in the following definition, this procedure can be completely hidden in the protocol specification by definition of the action **checksig**, where *seed* is a fresh variable, $seed \notin fv(P_1)$.

```
checksig sig pubkey T  $P_1$ else $P_2$ ≡
  case pubkey of PubKey(; seed)
    case sig of Sign(PrivKey(seed), T; ) $P_1$ else $P_2$
  else
    $P_2$
```

This style of specification can be applied analogously to asymmetric encryption and, simpler, symmetric encryption as is shown with the following definitions.

```
asymdec msg privkey content  $P_1$ else $P_2$ ≡
  case privkey of PrivKey(; seed)
    case msg of AsymEnc(PubKey(seed); content) $P_1$ else $P_2$
  else
    $P_2$

symdec msg symkey content  $P_1$ else $P_2$ ≡
  case msg of SymEnc(symkey; content) $P_1$ else $P_2$
```

For a Dolev-Yao style attacker specification, one will then equip the attacker with **asymdec** and **symdec** and the ability to apply the functions Sign, AsymEnc, and SymEnc, but not the function PrivKey.

2.4. Behavioural equivalences

In this section, we define several notions of equivalences for networks.

Definition 2.6 ($\mathcal{T}$-*Bisimilarity*). The relation $\mathcal{S}$ is called a $\mathcal{T}$-*simulation* if $N\,\mathcal{S}\,M$ implies: whenever $N \xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}} N'$ then, for some M', $M \xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}} M'$ and $N'\,\mathcal{S}\,M'$. $\mathcal{S}$ is called a $\mathcal{T}$-*bisimulation* if both $\mathcal{S}$ and its converse are $\mathcal{T}$-simulations. $\mathcal{T}$-*bisimilarity*, written $\sim_{\mathcal{T}}$, is the largest $\mathcal{T}$-bisimulation.

$\mathcal{T}$-bisimilarity allows for reasoning about networks on specific topologies as the following two examples show. As usual, since $\sim_{\mathcal{T}}$ is equal to the union of all $\mathcal{T}$-bisimulations, $\mathcal{T}$-bisimilarity of two networks N and M is shown by defining a set $\mathcal{S} \subseteq \mathbf{N} \times \mathbf{N}$ with $(N, M) \in \mathcal{S}$ and proving that it is a $\mathcal{T}$-bisimulation.

Example 2.7.

(1) Let $\mathcal{T}$ be a network topology isolating n, i.e. $\forall\, G \in \mathcal{T}.\ \nexists m.\ (m, n) \in E(G)$. Then,

$$n[\textbf{in}\ x.\,P, S]\ \sim_{\mathcal{T}} n[\textbf{nil}, S]\ .$$

(2) If $G \in \mathcal{T}$ implies $(n, m) \in E(G) \Leftrightarrow (n', m) \in E(G)$ for all $m \in V(G)$, then

$$n[\mathsf{in}\ x.\,\mathsf{nil}, S] \parallel n'[\mathsf{in}\ x.\,\mathsf{nil}, S] \sim_{\mathcal{T}} n[\mathsf{in}\ x.\,\mathsf{nil}, S]\ .$$

Proof. (1) Let $\mathcal{S} = \{(n[\mathsf{in}\ x.\,P, S], n[\mathsf{nil}, S])\}$. Since n is isolated, In$_2$ is the only rule applicable to $n[\mathsf{in}\ x.\,P, S]$ and $n[\mathsf{in}\ x.\,P, S] \xrightarrow{(U,m):}_{G\in\mathcal{T}} n[\mathsf{in}\ x.\,P, S]$ is the only derivation we can assume. $n[\mathsf{nil}, S]$ can simulate this with Nil: $n[\mathsf{nil}, S] \xrightarrow{(U,m):}_{G\in\mathcal{T}} n[\mathsf{nil}, S]$. The converse direction is analogous.

(2) Take $\mathcal{S} = \{P \in \{\mathsf{in}\ x.\,\mathsf{nil}, \mathsf{nil}\}\ :\ (n[P, S] \parallel n'[P, S], n[P, S])\}$. □

If networks are sought to be equivalent on *any* given network topology, one has to resort to the following definition.

Definition 2.8 (*Bisimilarity*). Networks N and M are said to be bisimilar, written $N \sim M$, if they are $\mathcal{T}_{\max}$-bisimilar.

Recall from Section 2.2 that $\mathcal{T}_{\max}$ contains all graphs on $\mathcal{N}_{loc}$.

The examples show that terminated nodes or nodes which no longer interact with the environment are insignificant with respect to network interaction.

Example 2.9.

(1) $n[\mathsf{nil}, \varepsilon] \parallel N \sim N$.
(2) If $n \notin V(N)$ then $n[\mathsf{nil}, S] \parallel N \sim N$.
(3) $n[\mathsf{read}\ x.\,\mathsf{nil}, S] \sim n[\mathsf{nil}, S]$.
(4) $n[\mathsf{store}\ T.\,\mathsf{nil}, S] \sim n[\mathsf{nil}, S]$.
(5) $n[\mathsf{nil}, S] \sim m[\mathsf{nil}, S]$.

Again, these propositions are proved by finding an appropriate $\mathcal{T}$-bisimulation $\mathcal{S}$ in each case. Note for (2) that $n \notin V(N)$ prevents N from acquiring yet unknown terms from S which could be sent and thus distinguish the networks.

The presented notion of bisimilarity distinguishes networks by their communication capabilities.

- Whenever a node n of network N sends, i.e. label $(U, m)!$, a node with the same name n in network M is also ready to send.
- Whenever one or more nodes in network N receive, i.e. label $(U, m)?$, one or more nodes in the network M will also receive.
- Whenever the complete network N loses a term, i.e. label $(U, m):$, all the nodes of the network M lose the term as well.

Internal actions such as storage and retrieval are ignored as long as they do not interfere with the communication capabilities, as shown in Example 2.9 (3) and (4). However, $n[\mathsf{store}\ T.\,P, S] \not\sim n[P, S]$ for arbitrary P if $T \notin S$, since $(T, n)!$ can distinguish them.

In order to distinguish networks by their capability to store terms, without having to rely on the existence of distinguishing communication actions, we define a barbed equivalence. The *barb predicate* $N \downarrow_n U$ (defined in Table 5) holds if N can "immediately" store term U at location n, i.e. without requiring another network interaction.

Table 5
Barb predicate

Barb-Empty	$N \downarrow_n \varepsilon$
Barb-Store	$n[\mathsf{store}\ T.\,P, S] \downarrow_n T$
Barb-Struct	$\dfrac{N \equiv N' \qquad N' \downarrow_n U}{N \downarrow_n U}$
Barb-PPar	$\dfrac{N \downarrow_n U}{N \parallel M \downarrow_n U}$

Table 6
Convergence predicate

CONV-BARB	$\dfrac{N \downarrow_n U}{N \Downarrow_n^{\mathcal{T}} U}$
CONV-COMM	$\dfrac{N \xrightarrow{(U,m)!}_{G\in\mathcal{T}} N' \qquad N' \Downarrow_n^{\mathcal{T}} U}{N \Downarrow_n^{\mathcal{T}} U}$

Definition 2.10 (*Barbed equivalence*). A relation $\mathcal{S}$ is called a *barbed simulation* if whenever $(N, M) \in \mathcal{S}$

(1) $N \downarrow_n U$ implies $M \downarrow_n U$ for each barb U and $n \in V(N) \cap V(M)$.

(2) $N \xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}_{max}} N'$ implies $M \xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}_{max}} M'$ for some M' and $(N', M') \in \mathcal{S}$.

$\mathcal{S}$ is called a *barbed bisimulation* if both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are barbed simulations. *Barbed equivalence*, written $\dot{\sim}$, is the largest barbed bisimulation.

As the following example shows, barbed equivalence will distinguish some networks which before were identified by bisimilarity.

Example 2.11. $n[\mathbf{store}\ T.\,\mathbf{nil}, S] \ \dot{\not\sim}\ n[\mathbf{nil}, S]$.

More generally, the following results can be checked by examining the definitions of the equivalences:

Theorem 2.12.

(1) *$N \sim M$ implies $N \sim_{\mathcal{T}} M$ for any $\mathcal{T}$.*

(2) *$N \dot{\sim} M$ implies $N \sim M$.*

Proof. (1) Fix a topology $\mathcal{T}$ and assume $N \sim M$. By definition of bisimilarity, there exists a $\mathcal{T}_{\max}$-bisimulation $\mathcal{S}$ such that if $N\,\mathcal{S}\,M$ and $N \xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}_{\max}} N'$ then, for some M', $M \xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}} M'$ and $N'\,\mathcal{S}\,M'$. Because $\mathcal{T} \subseteq \mathcal{T}_{\max}$, $\mathcal{S}$ is also a $\mathcal{T}$-bisimulation.

(2) By definition, barbed equivalence provides a $\mathcal{T}_{\max}$-bisimulation $\mathcal{S}$. □

However, it turns out that barbed equivalence is too fine grained to be useful for security analysis. This is because the desired security property (defined later in Section 4.3) only regards storage actions as crucial for security because they describe a long-term commitment of a node (an item put in a routing table will be used again and again); it does not matter on the other hand which messages are transmitted on the network (a secure protocol will just discard forged messages). For this the *convergence predicate* $N \Downarrow_n^{\mathcal{T}} U$ is defined in Table 6 and holds if N will eventually store U (possibly after some interactions under network topology $\mathcal{T}$) at location n.

The barb and convergence predicates observe terms which can be understood as (structured) data that can be passed around in networks. Naturally, some kinds of data are irrelevant for particular observations one wants to make, while others are crucial. This is the rationale behind the introduction of *mediation*, which allows us to focus on particular kinds of data.

Definition 2.13. A *mediator* μ is a function on terms $\mu : \mathbf{T} \to \mathbf{T}$. A mediator μ is called *simple* if $\mu(U) \in \{\varepsilon, U\}$ for all U.

The combination of mediator and convergence predicate then gives rise to the desired notion of equivalence:

Definition 2.14 (*Mediated equivalence*). For given topology $\mathcal{T}$ and mediator μ, we write $N \sqsubseteq^{\mu}_{\mathcal{T}} M$ whenever

$$N \Downarrow^{\mathcal{T}}_{n} U \text{ implies } M \Downarrow^{\mathcal{T}}_{n} \mu(U) \text{ for each barb } U \text{ and } n \in V(N) \cap V(M).$$

Mediated equivalence, written $\simeq^{\mu}_{\mathcal{T}}$, is then defined as follows:

$$N \simeq^{\mu}_{\mathcal{T}} M \quad \text{iff} \quad N \sqsubseteq^{\mu}_{\mathcal{T}} M \text{ and } M \sqsubseteq^{\mu}_{\mathcal{T}} N.$$

The following result relates barbed and mediated equivalence:

Theorem 2.15. *$N \stackrel{\bullet}{\sim} M$ implies $N \simeq^{id}_{\mathcal{T}} M$, where id represents the identity.*

Proof. We show $N \sqsubseteq^{id}_{\mathcal{T}} M$. The other inclusion follows from symmetry of $\stackrel{\bullet}{\sim}$. By definition of $\sqsubseteq^{id}_{\mathcal{T}}$ we have to show that for any term U and for all $n \in V(N) \cap V(M)$: $N \Downarrow^{\mathcal{T}}_{n} U$ implies $M \Downarrow^{\mathcal{T}}_{n} U$. Thus, we fix U and n and show the following by induction on the inference of $N \Downarrow^{\mathcal{T}}_{n} U$:

$$\text{If } N \Downarrow^{\mathcal{T}}_{n} U \text{ and } N \stackrel{\bullet}{\sim} M \text{ then } M \Downarrow^{\mathcal{T}}_{n} U$$

Case Conv-Barb: Because $N \Downarrow^{\mathcal{T}}_{n} U$ is due to Conv-Barb, $N \downarrow_{n} U$ holds. Because $N \stackrel{\bullet}{\sim} M$ we have $M \downarrow_{n} U$ by Definition 2.10. With rule Conv-Barb we have $M \Downarrow^{\mathcal{T}}_{n} U$.

Case Conv-Comm: Because $N \Downarrow^{\mathcal{T}}_{n} U$ is due to Conv-Comm, $N \xrightarrow{(U,m)!}_{G\in\mathcal{T}} N'$ and $N' \Downarrow^{\mathcal{T}}_{n} U$ hold. Because $N \stackrel{\bullet}{\sim} M$ we have that there exists M' such that $M \xrightarrow{(U,m)!}_{G\in\mathcal{T}} M'$ $(*)$ and $N' \stackrel{\bullet}{\sim} M'$. We can thus apply the induction hypothesis to have $M' \Downarrow^{\mathcal{T}}_{n} U$. With $(*)$ and Conv-Comm we have $M \Downarrow^{\mathcal{T}}_{n} U$. □

The mediator $\mu : \mathbf{T} \to \mathbf{T}$ is needed because the storage of some terms can be considered secure. It is used to fine tune the equivalence to the respective protocol specification. The next example illustrates this.

Example 2.16. Let a mediator μ_S and two networks N and $M(U)$ be defined as follows:

$$\begin{aligned} \mu_S(U) &= \begin{cases} \varepsilon & \text{if } U = T_{\text{sec}} \\ U & \text{otherwise} \end{cases} \\ N &\stackrel{\text{def}}{=} n[\mathbf{in}\ x.\, \mathbf{store}\ x.\, \mathbf{nil}, S] \\ M(U) &\stackrel{\text{def}}{=} m[\mathbf{out}\ U.\, \mathbf{nil}, S'] \end{aligned}$$

Then, the following holds because $N \Downarrow^{\mathcal{T}}_{n} \mu_S(T_{\text{sec}})$:

$$N \parallel M(T_{\text{sec}}) \simeq^{\mu_S}_{\mathcal{T}} N$$

However, the same is not true if the "secure" term T_{sec} is replaced by any other term T_{attack}.

Finally, the following theorem turns out to be helpful, and can be directly proved from the definition of $\sqsubseteq^{\mu}_{\mathcal{T}}$ and rule PPar:

Theorem 2.17. *For networks N, M and simple mediator μ, the following holds*:

$$N \sqsubseteq^{\mu}_{\mathcal{T}} N \parallel M$$

Proof. By definition of $\sqsubseteq^{\mu}_{\mathcal{T}}$, where we note that $V(N) \cap V(N \parallel M) = V(N)$, we have to show the following result:

$$\forall\, U.\, \forall\, n \in V(N).\ N \Downarrow^{\mathcal{T}}_{n} U \Rightarrow N \parallel M \Downarrow^{\mathcal{T}}_{n} \mu(U)$$

Fix thus $U \neq \varepsilon$ and $n \in V(N)$, and assume $N \Downarrow^{\mathcal{T}}_{n} U$. Because μ is simple (see Definition 2.13), we can do a case distinction on the value of $\mu(U)$. If $\mu(U) = \varepsilon$, we can show $N \parallel M \Downarrow^{\mathcal{T}}_{n} \mu(U)$ to hold directly with Conv-Barb and Barb-Empty.

Assume thus $\mu(U) \neq \varepsilon$, i.e. $\mu(U) = U$, since μ is simple. By induction on the shape of the derivation tree for $N \Downarrow_n^{\mathcal{T}} U$ (only examination of Conv-Comm and Conv-Barb are required) we know that there exists N' such that $N \longrightarrow_{\mathcal{T}}^{*} N'$ and $N' \downarrow_n U$ $(*)$.

We show the following auxiliary result by induction on the length of the derivation sequence for N':

$$N \longrightarrow_{\mathcal{T}}^{k} N' \Rightarrow \exists\, M'.\, N \parallel M \longrightarrow_{\mathcal{T}}^{k} N' \parallel M'$$

For $k = 0$ the result holds vacuously. We assume that $N \longrightarrow_{\mathcal{T}}^{k+1} N'$, which can be written as $N \longrightarrow_{\mathcal{T}}^{k} N'' \xrightarrow{(U,m)!}_{G\in\mathcal{T}} N'$. By induction hypothesis, there exists M'' such that $N \parallel M \longrightarrow_{\mathcal{T}}^{k} N'' \parallel M''$. By structural induction on M'', it can be shown that there exists M' such that $M'' \xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}} M'$ for either $\sharp = ?$ or $\sharp = :$. Using PPar on $N'' \xrightarrow{(U,m)!}_{G\in\mathcal{T}} N'$ and $M'' \xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}} M'$ we can establish the auxiliary result.

To establish the main result, take M' such that $N \parallel M \longrightarrow_{\mathcal{T}}^{*} N' \parallel M'$ $(**)$, where M' exists because of the auxiliary result. From $(*)$ and Barb-PPar we have $N' \parallel M' \downarrow_n U$, and with Conv-Barb also $N' \parallel M' \Downarrow_n^{\mathcal{T}} U$. By finitely many applications of Conv-Comm on $(**)$ we have $N \parallel M \Downarrow_n^{\mathcal{T}} U$. □

3. Control flow analysis

Control flow analysis is a program analysis technique to statically predict safe and computable approximations to the sets of values which may arise during program execution. While this technique was originally developed for functional languages [30], it has since then been applied to a variety of programming paradigms, including calculi for concurrency and security [6,5].

The result of our analysis for a network N yields an overapproximation of

(1) the set of terms which may be transmitted in N, together with their senders, and

(2) the set of terms which may be stored in N, together with the location of the storage.

This will enable us later (see Section 4) to automatically check whether networks are mediated equivalent and prove a security property for network protocols specified in CBS$^\sharp$.

In this section, we will first describe a static abstraction of the network topology $\mathcal{T}$, an important step to limit the state space arising from network execution. While our abstraction is simple, it retains the important properties we need for our security analysis. We then specify and describe our analysis and prove its semantic correctness by a subject reduction theorem and two corollaries relating the network actions sending and storage to our analysis result. We conclude with a brief overview of our implementation.

3.1. Topology abstractions

The evolution of a network N to a network N', formally expressed as $N \longrightarrow_{\mathcal{T}}^{*} N'$, implies that there is a sequence of graphs $G_1, G_2, \ldots, G_k \in \mathcal{T}$ such that

$$N \xrightarrow{(U_1,m_1)!}_{G_1} N_1 \xrightarrow{(U_2,m_2)!}_{G_2} N_2 \cdots \xrightarrow{(U_k,m_k)!}_{G_k} N'$$

and the graphs influence these derivations via rules In$_1$ and In$_2$. Thus, in order to overapproximate behaviour which might arise in the network, all *possible* links between senders and receivers have to be considered. Rather than considering all $G \in \mathcal{T}$ at any given step which would render infeasible the computation of the analysis, we can be safe by defining a static abstraction $\mathcal{G}(\mathcal{T})$ for $\mathcal{T}$ in the following way:

$$\mathcal{G}(\mathcal{T}) = \left(\bigcup_{G\in\mathcal{T}} V(G),\ \bigcup_{G\in\mathcal{T}} E(G)\right).$$

This means that an *abstract network topology* $\mathcal{G}(\mathcal{T})$ is again a connectivity graph, and contains all $G \in \mathcal{T}$ as subgraphs. It also ensures that an analysis over the abstract network topology will enable rule In$_1$ whenever a $G \in \mathcal{T}$ would have done it, since

$$(m, n) \in E(\mathcal{G}(\mathcal{T})) \text{ iff } \exists\, G \in \mathcal{T}.\ (m, n) \in E(G). \tag{1}$$

On the other hand, the analysis will always be safe with respect to rule IN_2 because IN_2 does not lead to the execution of an action.

3.2. Specification

We specify the analysis by a *Flow Logic* [22,20], which is an approach to static analysis that separates the specification of the acceptability of an analysis estimate from its computation. A flow logic specification consists of rules defining a judgement which expresses the relation of estimates and program fragments. The rules have to be interpreted co-inductively in the sense that an estimate is acceptable if it does not violate the conditions outset in the rules.

In our case, we define a syntax-directed analysis with the two judgements to range over the different syntactic categories:

$(\kappa, \sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} P$ judgement for processes
$(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$ judgement for networks

The main *judgement for networks* $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$ reads "(κ, σ) is a valid analysis estimate describing the behaviour of N under abstract network topology $\mathcal{G}(\mathcal{T})$". It is parametrised with $\mathcal{G}(\mathcal{T})$ and yields the following sets of values:

$\kappa \subseteq \mathbf{T} \times \mathcal{N}_{loc}$ network cache
$\sigma \subseteq \mathbf{T} \times \mathcal{N}_{loc}$ store cache

The contents of network and store cache can be intuitively described with the following statements which hold during execution of network N.

(1) If the term T may be sent from location n, then $(T, n) \in \kappa$.
(2) If the term T may be stored at location n, then $(T, n) \in \sigma$.

The *judgement for processes* $(\kappa, \sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} P$ is furthermore parametrised with the location n at which the particular process P is running, and carries the following local environment:

$\theta : \mathcal{X} \to \mathbf{T}$ substitution environment

Again intuitively, if $\theta(x) = U$ the term U may be bound to variable x during execution of P. By abuse of notation, we write $[U/x]$ to denote the binding of U to x and also use the substitution environment like a substitution in the sense of Section 2.1. The empty substitution environment is denoted $[]$. Note that the use of a local environment allows for a more precise analysis of the case statement in our language than in our previous work [19] or of similar constructs in LySa [5]. This proves to be crucial for limiting the number of false positives arising from the security analysis. As a trade-off, there is an increase in the complexity of the analysis. We show in [17] that it holds for both analysis variants that, given a protocol P run by the nodes of a network N, the worst-case complexity of analysing N is governed by a polynomial in the size of the universe of values with a degree independent of the size of N. However, this degree is higher if a local environment is used because it depends additionally on the depth of P and not only on the maximum arity of functions used in P (as it can be shown in the case of a global environment).

After this overview of the judgements, we turn to the formal specification of the analysis in Table 7 and explain the rules in the following.

Judgement for networks: Rule CFA-NODE says that (κ, σ) is a valid analysis result describing the behaviour of node $n[P[U_i/x_i]_{i=1}^{k}, S]$ under abstract connectivity graph $\mathcal{G}(\mathcal{T})$ iff it is also a valid analysis result for process P at n with a fresh variable environment which only contains bindings corresponding to the substitutions $[U_i/x_i]_{i=1}^{k}$ P might carry, and all terms contained in store S are element of σ at n. Rule CFA-PPAR is straightforward.

Judgement for processes: Rule CFA-NIL is straightforward. Rule CFA-OUT says that the analysis of process **out** $T.P$ is achieved by the following: updating the network cache κ with terms $(T\theta, n)$ and computing the analysis for the continuation process P.

Rule CFA-IN on the other hand looks at all terms (U, m) recorded in κ. For an edge $(m, n) \in E(\mathcal{G}(\mathcal{T}))$, the local variable environment is updated at x with U and continuation P evaluated under this new environment.

Rule CFA-STORE is similar to CFA-OUT. However, instead of inserting a term into the network cache, the insertion is into the store cache σ at n, $(T\theta, n) \in \sigma$. On the other hand, rule CFA-READ resembles rule CFA-IN: terms are now

Table 7
Control flow analysis for CBS$^\sharp$

Judgement for Processes

CFA-Nil $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} \textsf{nil}$

CFA-Out $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} \textsf{out}\ T.\ P$
iff $(T\theta, n) \in \kappa \wedge (\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} P$

CFA-In $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} \textsf{in}\ x.\ P$
iff $\forall\, (U,m) \in \kappa.\ (m,n) \in E(\mathcal{G}(\mathcal{T})) \Rightarrow (\kappa,\sigma) \models^{\theta[U/x]}_{\mathcal{G}(\mathcal{T}),n} P$

CFA-Store $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} \textsf{store}\ T.\ P$
iff $(T\theta, n) \in \sigma \wedge (\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} P$

CFA-Read $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} \textsf{read}\ x.\ P$
iff $\forall\, (U,n) \in \sigma.\ (\kappa,\sigma) \models^{\theta[U/x]}_{\mathcal{G}(\mathcal{T}),n} P$

CFA-Case $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} \textsf{case}\ T\ \textsf{of}\ f(T_1 \dots T_j; x_{j+1} \dots x_k)\ P_1\ \textsf{else}\ P_2$
iff $(T\theta = f(V_1 \dots V_k) \wedge \bigwedge_{i=1}^{j} T_i\theta = V_i \Rightarrow (\kappa,\sigma) \models^{\theta[V_i/x_i]_{i=j+1}^{k}}_{\mathcal{G}(\mathcal{T}),n} P_1) \wedge$
$(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} P_2$

CFA-Par $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} P_1 \mid P_2$
iff $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} P_1 \wedge (\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} P_2$

CFA-Repl $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} !P$
iff $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} P$

Judgement for Networks

CFA-Node $(\kappa,\sigma) \models_{\mathcal{G}(\mathcal{T})} n[P\theta, S]$
iff $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\mathcal{T}),n} P \wedge \forall\, U \in S.\ (U,n) \in \sigma$

CFA-PPar $(\kappa,\sigma) \models_{\mathcal{G}(\mathcal{T})} N_1 \parallel N_2$
iff $(\kappa,\sigma) \models_{\mathcal{G}(\mathcal{T})} N_1 \wedge (\kappa,\sigma) \models_{\mathcal{G}(\mathcal{T})} N_2$

taken from the store cache instead of the network cache, and the continuation P is evaluated under the updated variable environment.

In rule CFA-Case, if $T\theta$ is of the form $f(V_1 \dots V_k)$, it is checked that $T_1\theta = V_1 \wedge \dots \wedge T_j\theta = V_j$, meaning that the first j arguments of f match. Then the continuation P_1 is analysed under a substitution environment which maps x_i to V_i for $i = j+1, \dots, k$, meaning the remaining arguments of f are bound to the variables x_i.

Rule CFA-Par and rule CFA-Repl are straightforward.

3.3. Semantic correctness

In Section 3.2 we have informally stated what elements a valid analysis estimate (κ,σ) will contain. The goal of this section is to formally establish these statements. Classically, correctness of the analysis is shown by proving three desirable properties: well-definedness of the judgements to ensure that the functional they define has fixpoints; semantic correctness of the judgements; Moore family property to ensure that the analysis has a most precise result. Well-definedness and the Moore family property are straightforward using the techniques described in [20] and therefore not included in this paper. Semantic correctness with respect to the operational semantics of Section 2.2 is stated as a subject reduction theorem and proved in this section. Informally, it expresses that the analysis estimate remains acceptable when the network evolves. From this, statements about κ and σ follow directly.

The following lemma states auxiliary subject reduction results which hold for the reduction relation and structural equivalence.

Lemma 3.1.
(1) *If* $N \rightarrow N'$ *and* $(\kappa, \sigma) \models_{\widehat{G}} N$, *then* $(\kappa, \sigma) \models_{\widehat{G}} N'$.
(2) *If* $N \equiv N'$ *and* $(\kappa, \sigma) \models_{\widehat{G}} N$, *then* $(\kappa, \sigma) \models_{\widehat{G}} N'$.

Proof. (1) The proof is by induction on the inference of $N \rightarrow N'$.

Case Store: Then $N = n[\textbf{store}\ T.\ P, S]$ and $N' = n[P, S \cup T]$.

$$\begin{array}{lr}
(\kappa, \sigma) \models_{\widehat{G}} n\ [\textbf{store}\ T.\ P, S] & \text{(by assumption)} \\
\text{thus } (\kappa, \sigma) \models^{[]}_{\widehat{G},n} \textbf{store}\ T.\ P \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Node)} \\
\text{thus } (T, n) \in \sigma \wedge (\kappa, \sigma) \models^{[]}_{\widehat{G},n} P \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Store)} \\
\text{thus } (\kappa, \sigma) \models^{[]}_{\widehat{G},n} P \wedge \forall\, U \in S \cup T.\ (U, n) \in \sigma & \\
\text{thus } (\kappa, \sigma) \models_{\widehat{G}} n\ [P, S \cup T] & \text{(by CFA-Node)}
\end{array}$$

Case Read: Then $N = n[\textbf{read}\ x.\ P, S]$ and $N' = n[P[T/x], S]$. From the preconditions of Read we know $T \in S$ $(*)$.

$$\begin{array}{lr}
(\kappa, \sigma) \models_{\widehat{G}} n\ [\textbf{read}\ x.\ P, S] & \text{(by assumption)} \\
\text{thus } (\kappa, \sigma) \models^{[]}_{\widehat{G},n} \textbf{read}\ x.\ P \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Node)} \\
\text{thus } \forall\, (U, n) \in \sigma.\ (\kappa, \sigma) \models^{[U/x]}_{\widehat{G},n} P \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Read)} \\
\text{thus } (\kappa, \sigma) \models^{[T/x]}_{\widehat{G},n} P \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by } (*)) \\
\text{thus } (\kappa, \sigma) \models_{\widehat{G}} n\ [P[T/x], S] & \text{(by CFA-Node)}
\end{array}$$

Case Case$_1$: Then $N = n[\textbf{case}\ T\ \textbf{of}\ f(\widetilde{T}; \widetilde{x})\ P_1\ \textbf{else}\ P_2, S]$ and $N' = n[P_1[\widetilde{U}/\widetilde{x}], S]$, where $T = f(\widetilde{T}, \widetilde{U})$ $(*)$ from the preconditions of Case$_1$ and $\widetilde{T} = T_1, \ldots, T_j, \widetilde{x} = x_{j+1}, \ldots, x_k, \widetilde{U} = U_{j+1}, \ldots, U_k$.

$$\begin{array}{lr}
(\kappa, \sigma) \models_{\widehat{G}} n\ [\textbf{case}\ T\ \textbf{of}\ f(\widetilde{T}; \widetilde{x})\ P_1\ \textbf{else}\ P_2, S] & \text{(by assumption)} \\
\text{thus } (\kappa, \sigma) \models^{[]}_{\widehat{G},n} \textbf{case}\ T\ \textbf{of}\ f(\widetilde{T}; \widetilde{x})\ P_1\ \textbf{else}\ P_2 \wedge & \\
\quad \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Node)} \\
\text{thus } (T = f(V_1, \ldots, V_k) \wedge \bigwedge_{i=1}^{j} T_i = V_i \Rightarrow & \\
\quad (\kappa, \sigma) \models^{[V_i/x_i]_{i=j+1}^{k}}_{\widehat{G},n} P_1) \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Case)} \\
\text{thus } (\kappa, \sigma) \models^{[U_i/x_i]_{i=j+1}^{k}}_{\widehat{G},n} P_1 \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by } (*)) \\
\text{thus } (\kappa, \sigma) \models_{\widehat{G}} n\ [P_1[\widetilde{U}/\widetilde{x}], S] & \text{(by CFA-Node)}
\end{array}$$

Case Case$_2$: Then $N = n[\textbf{case}\ T\ \textbf{of}\ f(\widetilde{T}; \widetilde{x})\ P_1\ \textbf{else}\ P_2, S]$ and $N' = n[P_2, S]$.

$$\begin{array}{lr}
(\kappa, \sigma) \models_{\widehat{G}} n\ [\textbf{case}\ T\ \textbf{of}\ f(\widetilde{T}; \widetilde{x})\ P_1\ \textbf{else}\ P_2, S] & \text{(by assumption)} \\
\text{thus } (\kappa, \sigma) \models^{[]}_{\widehat{G},n} \textbf{case}\ T\ \textbf{of}\ f(\widetilde{T}; \widetilde{x})\ P_1\ \textbf{else}\ P_2 \wedge & \\
\quad \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Node)} \\
\text{thus } (\kappa, \sigma) \models^{[]}_{\widehat{G},n} P_2 \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Case)} \\
\text{thus } (\kappa, \sigma) \models_{\widehat{G}} n\ [P_2, S] & \text{(by CFA-Node)}
\end{array}$$

Case Repl$_2$: Then $N = n[!P, S]$ and $N' = n[P, S]$.

$$\begin{array}{lr}
(\kappa, \sigma) \models_{\widehat{G}} n[!P, S] & \text{(by assumption)} \\
\text{thus } (\kappa, \sigma) \models_{\widehat{G}} n[P, S] & \text{(by CFA-Repl)}
\end{array}$$

(2) The proof is by induction on the inference of $N \equiv N'$.

Case Comm: Then $N = N_1 \parallel N_2$ and $N' = N_2 \parallel N_1$.

$$\begin{array}{rll} & (\kappa, \sigma) \models_{\widehat{G}} N_1 \parallel N_2 & \text{(by assumption)} \\ \text{thus} & (\kappa, \sigma) \models_{\widehat{G}} N_1 \wedge (\kappa, \sigma) \models_{\widehat{G}} N_2 & \text{(by CFA-PPar)} \\ \text{thus} & (\kappa, \sigma) \models_{\widehat{G}} N_2 \parallel N_1 & \text{(by CFA-PPar)} \end{array}$$

Case Assoc: Then $N = N_1 \parallel (N_2 \parallel N_3)$ and $N' = (N_1 \parallel N_2) \parallel N_3$. The result is established by four applications of CFA-PPar analogously to case Comm.

Case Refl: Then $N = N'$. Nothing to show.

Case Sym: We have $(\kappa, \sigma) \models_{\widehat{G}} N$ by assumption. The induction hypothesis applied to $N' \equiv N$ is

$$N' \equiv N \wedge (\kappa, \sigma) \models_{\widehat{G}} N' \Rightarrow (\kappa, \sigma) \models_{\widehat{G}} N \qquad \square$$

Since $(\kappa, \sigma) \models_{\widehat{G}} N$ and $N' \equiv N$ are true, $(\kappa, \sigma) \models_{\widehat{G}} N'$ must be true as well.

Case Trans: We have $(\kappa, \sigma) \models_{\widehat{G}} N$ by assumption and can apply the induction hypothesis first to $N \equiv N''$ to get $(\kappa, \sigma) \models_{\widehat{G}} N''$, and thus a second time to $N'' \equiv N'$ to get $(\kappa, \sigma) \models_{\widehat{G}} N'$.

Case Comp: Then $N = N_1 \parallel N_2$ and $N' = N_1' \parallel N_2$. We have $(\kappa, \sigma) \models_{\widehat{G}} N_1 \parallel N_2$ and with CFA-PPar also $(\kappa, \sigma) \models_{\widehat{G}} N_1$ and $(\kappa, \sigma) \models_{\widehat{G}} N_2$. We can thus apply the induction hypothesis to $N_1 \equiv N_1'$ to get $(\kappa, \sigma) \models_{\widehat{G}} N_1'$. Together with $(\kappa, \sigma) \models_{\widehat{G}} N_2$ we have $(\kappa, \sigma) \models_{\widehat{G}} N_1' \parallel N_2$ as desired with CFA-PPar.

Case Red: We have $(\kappa, \sigma) \models_{\widehat{G}} N$ by assumption and $N \rightarrow N'$ by Red and can apply Lemma 3.1 (1) to yield $(\kappa, \sigma) \models_{\widehat{G}} N'$.

Case Par: Then $N = n[P_1 \mid P_2, S_1 \cup S_2]$ and $N' = n[P_1, S_1] \parallel n[P_2, S_2]$.

$$\begin{array}{rll} & (\kappa, \sigma) \models_{\widehat{G}} n\,[P_1 \mid P_2, S_1 \cup S_2] & \text{(by assumption)} \\ \text{thus} & (\kappa, \sigma) \models^{[]}_{\widehat{G},n} P_1 \mid P_2 \wedge \forall\, U \in S_1 \cup S_2.\ (U, n) \in \sigma & \text{(by CFA-Node)} \\ \text{thus} & (\kappa, \sigma) \models^{[]}_{\widehat{G},n} P_1 \wedge (\kappa, \sigma) \models^{[]}_{\widehat{G},n} P_2 \wedge & \\ & \quad \forall\, U \in S_1.\ (U, n) \in \sigma \wedge \forall\, U \in S_2.\ (U, n) \in \sigma & \text{(by CFA-Par)} \\ & (\kappa, \sigma) \models_{\widehat{G}} n\,[P_1, S_1] \parallel n[P_2, S_2] & \text{(by CFA-Node)} \quad \square \end{array}$$

The following lemma is needed in order to simplify the proofs of Theorem 3.3 (and later also Theorem 3.4). The lemma states that if network N can send term T from location n, then the message cache κ computed by an analysis of N contains (T, n).

Lemma 3.2. *If* $N \xrightarrow{(T,n)!}_{G \in \mathcal{T}} N'$ *and* $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$, *then* $(T, n) \in \kappa$.

Proof. The proof is by induction on the inference of $N \xrightarrow{(T,n)!}_{G \in \mathcal{T}} N'$. Considering the label $(T, n)!$, it suffices to distinguish the following three cases:

Case Out$_1$: Then $N = n[\textbf{out}\ T.\,P, S]$. We have $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} n\,[\textbf{out}\ T.\,P, S]$ by assumption. We can apply CFA-Node to have $(\kappa, \sigma) \models^{[]}_{\mathcal{G}(\mathcal{T}),n} \textbf{out}\ T.\,P$, and then CFA-Out to get $(T, n) \in \kappa$.

Case PPar: Then $N = N_1 \parallel N_2$. We have $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N_1 \parallel N_2$ by assumption, and with CFA-PPar thus $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N_1$ and $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N_2$. The premises of PPar are $N_1 \xrightarrow{(T,n)\sharp_1}_{G \in \mathcal{T}} N_1'$ and $N_2 \xrightarrow{(T,n)\sharp_2}_{G \in \mathcal{T}} N_2'$. We know $\sharp_1 \circ \sharp_2 = !$, hence either $\sharp_1 = !$ or $\sharp_2 = !$ by properties of $\circ$. For the premise labelled $(T, n)!$, together with either $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N_1$ or $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N_2$, whichever is relevant, the induction hypothesis can be applied to give $(T, n) \in \kappa$.

Case Struct: We have $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$ by assumption. We assume the premises of Struct, in particular $N \equiv M$ and $M \xrightarrow{(T,n)\sharp}_{G \in \mathcal{T}} M'$. Applying Lemma 3.1 (2) to $N \equiv M$ and $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$ gives $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} M$. Thus, the induction hypothesis can be applied to give $(T, n) \in \kappa$. $\square$

Using Lemmas 3.1 and 3.2, we can prove the main semantic correctness theorem, which ensures that the analysis estimate is a safe description of what will happen during the evolution of a network.

Theorem 3.3 (*Subject reduction*).

$$\text{If } N \xrightarrow{(U,m)!}_{G\in\mathcal{T}} N' \text{ and } (\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} N, \text{ then } (\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} N'.$$

Proof. The proof requires that we consider general labels $(U,m)\sharp$. We thus show the result by induction on the inference of $M \xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}} M'$, where M and M' are subterms of N and N', respectively.

Case Nil: Then $M = M' = n[\mathbf{nil}, S]$. Nothing to show.

Case Out$_1$: Then $M = n[\mathbf{out}\ T.\,P, S]$ and $M' = n[P, S]$. We have

$$\begin{array}{ll}
(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} n\ [\mathbf{out}\ T.\,P, S] & \text{(by assumption)}\\
\text{thus } (\kappa,\sigma)\models^{[]}_{\mathcal{G}(\mathcal{T}),n} \mathbf{out}\ T.\,P \wedge \forall\, U\in S.\ (U,n)\in\sigma & \text{(by CFA-Node)}\\
\text{thus } (\kappa,\sigma)\models^{[]}_{\mathcal{G}(\mathcal{T}),n} P \wedge \forall\, U\in S.\ (U,n)\in\sigma & \text{(by CFA-Out)}\\
\text{thus } (\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} n\ [P, S] & \text{(by CFA-Node)}
\end{array}$$

Case Out$_2$: Then $M = M' = n[\mathbf{out}\ T.\,P, S]$. Nothing to show.

Case In$_1$: Then $M = n[\mathbf{in}\ x.\,P, S]$ and $M' = n[P[U/x], S]$. By assuming $N \xrightarrow{(U,m)!}_{G\in\mathcal{T}} N'$ and $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} N$, Lemma 3.2 gives $(U,m)\in\kappa$ $(*)$. From the preconditions of In$_1$ we know $(m,n)\in E(G)$, and with Eq. (1) on 215 we have $(m,n)\in E(\mathcal{G}(\mathcal{T}))$ $(**)$.

$$\begin{array}{ll}
(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} n\ [\mathbf{in}\ x.\,P, S] & \text{(by assumption)}\\
\text{thus } (\kappa,\sigma)\models^{[]}_{\mathcal{G}(\mathcal{T}),n} \mathbf{in}\ x.\,P \wedge \forall\, U\in S.\ (U,n)\in\sigma & \text{(by CFA-Node)}\\
\text{thus } (\forall\ (U',m')\in\kappa.\ (m',n)\in E(\mathcal{G}(\mathcal{T}))\Rightarrow & \\
\qquad (\kappa,\sigma)\models^{[U'/x]}_{\mathcal{G}(\mathcal{T}),n} P) \wedge \forall\, U\in S.\ (U,n)\in\sigma & \text{(by CFA-In)}\\
\text{thus } ((m,n)\in E(\mathcal{G}(\mathcal{T}))\Rightarrow (\kappa,\sigma)\models^{[U/x]}_{\mathcal{G}(\mathcal{T}),n} P) \wedge & \\
\qquad \forall\, U\in S.\ (U,n)\in\sigma & \text{(by } (*))\\
\text{thus } (\kappa,\sigma)\models^{[U/x]}_{\mathcal{G}(\mathcal{T}),n} P \wedge \forall\, U\in S.\ (U,n)\in\sigma & \text{(by } (**))\\
\text{thus } (\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} n\ [P[U/x], S] & \text{(by CFA-Node)}
\end{array}$$

Case In$_2$: Then $M = M' = n[\mathbf{in}\ x.\,P, S]$. Nothing to show.

Case Repl$_1$: Then $M = M' = n[!P, S]$. Nothing to show.

Case PPar: Then $M = M_1 \parallel M_2$ and $M' = M_1' \parallel M_2'$. We have $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} M_1 \parallel M_2$ by assumption, and with CFA-PPar thus $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} M_1$ and $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} M_2$. We can use the induction hypothesis twice on $M_1 \xrightarrow{(U,m)\sharp_1}_{G\in\mathcal{T}} M_1'$ and $M_2 \xrightarrow{(U,m)\sharp_2}_{G\in\mathcal{T}} M_2'$ to have $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} M_1'$ and $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} M_2'$. With CFA-PPar we have $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} M_1' \parallel M_2'$.

Case Struct: We have $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} M$ by assumption. We assume the preconditions of Struct: $M \equiv L$, $L \xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}} L'$, and $L' \equiv M'$. By application of Lemma 3.1 (2) to $M \equiv L$ and $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} M$ we have $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} L$. Hence, we can apply the induction hypothesis on $L \xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}} L'$ to have $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} L'$. By applying Lemma 3.1 (2) again, we have $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} L'$. □

The two main results describing the contents of a valid analysis estimation almost directly follow from this subject reduction result. The first result formalises our previous claim "if the term T may be sent from location n during evolution of a network N, then $(T,n)\in\kappa$".

Theorem 3.4. *If* $N \longrightarrow^*_{\mathcal{T}} N' \xrightarrow{(T,n)!}_{G\in\mathcal{T}} N''$ *and* $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} N$, *then* $(T,n)\in\kappa$.

Proof. We have $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} N$ by assumption and can thus apply Theorem 3.3 finitely many times to the derivations $N \longrightarrow^*_{\mathcal{T}} N'$ to yield $(\kappa,\sigma)\models_{\mathcal{G}(\mathcal{T})} N'$. With $N' \xrightarrow{(T,n)!}_{G\in\mathcal{T}} N''$ we can apply Lemma 3.2 to have $(T,n)\in\kappa$. □

The next theorem formalises “if a term $T \neq \varepsilon$ may be stored at location n during evolution of a network N, then $(T, n) \in \sigma$”.

Theorem 3.5. *For $T \neq \varepsilon$, the following implication holds: if $N \Downarrow_n^{\mathcal{T}} T$* and *$(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$, then $(T, n) \in \sigma$.*

Proof. By induction on the inference of $N \Downarrow_n^{\mathcal{T}} T$, which involves only finitely many applications of Conv-Comm, Conv-Barb, Barb-PPar, Barb-Struct, and Barb-Store, there exist N', N'', P, S such that the following statements hold:

$$N \ \longrightarrow_{\mathcal{T}}^{*} N' \wedge N' \equiv N'' \parallel n[\textsf{store}\ T.\, P, S]\ .$$

Since we have $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$ by assumption, we can thus apply Theorem 3.3 finitely many times to the derivations $N \longrightarrow_{\mathcal{T}}^{*} N'$ to yield $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N'$. We can apply Lemma 3.1 (2) to have $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N'' \parallel n[\textsf{store}\ T.\, P, S]$. Furthermore, by CFA-PPar we have $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} n[\textsf{store}\ T.\, P, S]$, and we can apply CFA-Store as in the corresponding case of the proof of Lemma 3.1 (1) to have $(T, n) \in \sigma$. □

3.4. Implementation

We have implemented our control flow analysis using the *succinct solver* [21], a constraint solving tool. Constraints are specified as formulae in a Horn-like fragment of first-order predicate logic, called *alternation-free least fixed point logic* (ALFP), and the solver computes interpretations of predicates which satisfy those formulae. The analysis of Table 7 can almost directly be translated into a generation function for ALFP formulae, however, to achieve a finite universe of values we have to use regular grammars to represent terms, a technique introduced in [23]. The implementation is described in more detail in [17].

4. Security analysis of mobile wireless networks

In Section 2, we have presented a calculus particularly suitable to model the behaviour of mobile wireless networks and Section 3 established a control flow analysis on terms of this calculus to overapproximate the sets of terms transmitted and stored in a network. In this section we will show how these results can be combined into a framework for security analysis of mobile wireless networks.

We start by pointing out differences of secure communication protocols for mobile wireless networks from the more “traditional” security protocols for authentication, confidentiality and similar properties. We will then describe μSAODV, a simplified version of the combination of the routing protocol AODV [26] and its security extension SAODV [14]. μSAODV motivates the definition of a consistency condition for networks, a building block for routing security. The condition is based on the notion of mediated equivalence, developed in Section 2.4, and we show a result relating our analysis estimate to mediated equivalent networks. Our framework can thus be used for automated security analysis. We conclude the section with the analysis results for μSAODV showing that the protocol is insecure, and present a simple attack.

4.1. Security protocols vs. secure communication protocols

In Section 1.1 we have described the main operational characteristics of communication protocols for mobile wireless networking. Here, we want to clarify their security characteristics. For this purpose, it is helpful to compare them with the more common security protocols for authentication, confidentiality and similar properties. This is done in the following with respect to properties and modelling aspects:

Security properties: Security protocols are *designed* to achieve one or more specific security properties. In contrast, communication protocols provide network services which are not related to any kind of security property. Securing such protocols means to ensure that these network services can be provided unconditionally, even in an adversarial environment.

Security modelling: Security protocols are usually end-to-end, meaning that only the endpoints of communications are modelled and the environment is replaced by the Dolev-Yao attacker. For communication protocols, the states of the

Table 8
Subroutines of μSAODV specified in CBS$^\sharp$

```
$SendRREQ(dstip, ip) \overset{\text{def}}{=}$
  out RREQ($dstip, ip, ip$, PubKey($ip$),
          Sign(PrivKey($ip$), REQTuple($dstip, ip$, PubKey($ip$))))

$ReceiveRREQ(ip) \overset{\text{def}}{=}$
  in RREQ(; $dstip, origip, sndip, pubkey, sig$).
  case $pubkey$ of PubKey($origip$; )
    asymdec $sig$ $pubkey$ REQTuple($dstip, origip, pubkey$)
      store Route($origip, ip, sndip$).
      if $dstip = ip$ then
        out RREP($dstip, origip, sndip, ip$, PubKey($ip$),
                Sign(PrivKey($ip$), REPTuple($dstip, origip$, PubKey($ip$))))
      else
        out RREQ($dstip, origip, ip, pubkey, sig$)
    else nil
  else nil

$ReceiveRREP(ip) \overset{\text{def}}{=}$
  in RREP(; $dstip, origip, addrip, sndip, pubkey, sig$).
  if $addrip = ip$ then
    case $pubkey$ of PubKey($dstip$; )
      asymdec $sig$ $pubkey$ REPTuple($dstip, origip, pubkey$)
        store Route($dstip, ip, sndip$).
        if $origip = ip$ then
          nil
        else
          read Route($origip, ip$; $nexthop$)
            out RREP($dstip, origip, nexthop, ip, pubkey, sig$)
          else nil
      else nil
    else nil
  else nil
```

intermediate nodes and their connectivity matter and cannot be abstracted away. Consequentially, the attacker should be limited by the environment conditions as well.

Before applying these insights to the definition of a consistency condition for routing security in Section 4.3, we will now specify a concrete routing protocol for later analysis in our framework.

4.2. μSAODV

The AODV protocol [26], also standardised by the IETF as RFC 3561 [25], is a routing protocol for mobile ad hoc networks in which a node tries to find a route to a destination only if needed (on-demand). Its operation is based on routing tables and requires that each node stores a "vector" of direction (the next hop) and distance (number of hops) for a particular destination. SAODV [14], in the process of standardisation by the IETF [13], is a protocol extension of AODV to secure the route discovery mechanism.

We illustrate the use of our specification and analysis framework with μSAODV, a simplified version of the combination of the two protocols which describes only the route discovery step and no route maintenance and concentrates on the authentication mechanism of SAODV.

Operation: In Table 8 we show the main subroutines of μSAODV, modelled in CBS$^\sharp$. We describe the operation of the protocol by referring to this model.

If a node n_s (the *source*) needs to communicate with another node n_d (the *destination*) for which it has no routing information, n_s initiates a *route discovery* process. A route discovery comprises the following steps:

Sending RREQs: n_s initiates the route discovery by broadcasting a *route request RREQ* which contains the destination IP address, the source IP address, the IP address of the immediate sender, the public key of the source, and a signature of message type, destination IP, source IP, and source public key, with the private key of the source. In our notation, this amounts to the message $\mathsf{RREQ}(n_\mathrm{d}, n_\mathrm{s}, n_\mathrm{s}, \mathsf{PubKey}(n_\mathrm{s}), \mathit{sig})$, where *sig* is $\mathsf{Sign}(\mathsf{PrivKey}(n_\mathrm{s}), \mathsf{REQTuple}(n_\mathrm{d}, n_\mathrm{s}, \mathsf{PubKey}(n_\mathrm{s})))$.

Receiving RREQs: Every node n receiving a route request $\mathsf{RREQ}(n_\mathrm{d}, n_\mathrm{s}, n_\mathrm{i}, \mathit{pubkey}, \mathit{sig})$ will first check whether the provided public key belongs to the source. For this, the existence of a public-key infrastructure is assumed; this can be modelled elegantly by using the respective IP address as seed in the public/private key generation. It will then check whether the source signed the tuple $\mathsf{REQTuple}(n_\mathrm{d}, n_\mathrm{s}, \mathit{pubkey})$ with its private key. If one of these checks fails, n will abort. Otherwise, it makes an entry into its routing table to provide a *reverse route* leading to the source. The entry reads $\mathsf{Route}(n_\mathrm{s}, n, n_\mathrm{i})$ and means that whenever a packet addressed to n_s arrives at n, it will be forwarded to the next hop n_i, the immediate sender of the $RREQ$.

n will then check whether it is the destination itself, i.e. $n_\mathrm{d} = n$. If so, the n will send out a *route reply* $RREP$, containing its IP address, the source IP address, the IP address of the next hop of the reverse path (the addressee, in our case the immediate sender n_i), its public key, and a signature of the tuple of non-mutable fields with its private key. $\mathsf{RREP}(n_\mathrm{d}, n_\mathrm{s}, n_\mathrm{i}, n_\mathrm{d}, \mathsf{PubKey}(n_\mathrm{d}), \mathit{sig}')$, where sig' is $\mathsf{Sign}(\mathsf{PrivKey}(n_\mathrm{d}), \mathsf{REPTuple}(n_\mathrm{d}, n_\mathrm{s}, \mathsf{PubKey}(n_\mathrm{d})))$

If $n_\mathrm{d} \neq n$, n will rebroadcast the request, changing only the IP address of the immediate sender to its own IP: $\mathsf{RREQ}(n_\mathrm{d}, n_\mathrm{s}, n, \mathsf{PubKey}(n_\mathrm{s}), \mathit{sig})$.

Receiving RREPs: Every node n receiving a route reply message $\mathsf{RREP}(n_\mathrm{d}, n_\mathrm{s}, n_\mathrm{a}, n_\mathrm{i}, \mathit{pubkey}', \mathit{sig}')$ checks whether it is the addressee, i.e. $n_\mathrm{a} = n$. This implements a unicast on top of the broadcast, as all nodes will just drop the message if they are not addressed. Otherwise, n will check whether the provided public key belongs to the destination, and whether the destination signed $\mathsf{REPTuple}(n_\mathrm{d}, n_\mathrm{s}, \mathit{pubkey}')$. Again, n will abort on failure of any of these checks. Otherwise, it makes an entry into its routing table to provide a *forward route* leading to the destination, $\mathsf{Route}(n_\mathrm{d}, n, n_\mathrm{i})$.

If n was the initiator of the $RREQ$ in the first place, i.e. $n_\mathrm{s} = n$, it can now start sending data packets to n_d. Otherwise, n retrieves the route table entry $\mathsf{Route}(n_\mathrm{s}, n, n_\mathrm{x})$ for the reverse route to n_s from the store to get the next hop n_x on the reverse route. It then rebroadcasts the route reply as $\mathsf{RREP}(n_\mathrm{d}, n_\mathrm{s}, n_\mathrm{x}, n, \mathit{pubkey}', \mathit{sig}')$.

4.3. Security model for routing protocols

From the previous section, we can draw the following conclusions for routing table-based protocols such as μSAODV: every node is a reactive system in the sense that it offers a discrete interface to the environment and will accept and process any incoming message matching certain formats. It will thus accept messages of honest nodes and attackers alike. However, a node will only *commit* to this information if it updates its routing table accordingly. The ability of nodes to filter out malicious information at this stage determines the degree of security a protocol is offering. But when can such information be considered as malicious? A minimal requirement seems to be that the information should accurately represent the network topology. This leads to the definition of the following condition.

4.3.1. A consistency condition for routing networks

We define a consistency mediator $\mu_\mathcal{T} : \mathbf{T} \to \mathbf{T}$ such that $\mu_\mathcal{T}(U)$ evaluates to ε whenever the topology is “correctly represented” and to U otherwise. As different routing protocols will have different data representations for routing information, a formalisation of “correct representation” is only possible in the context of a particular protocol specification. This is shown here for μSAODV.

Example 4.1 (*Consistency mediator for μSAODV*).

$$\mu_\mathcal{T}^{\mathit{saodv}}(U) = \begin{cases} \varepsilon & \text{if for } U = \mathsf{Route}(n_d, n_1, n_2) \text{ there exist locations } n_3, \ldots, n_k \text{ such that } n_k = n_d \text{ and} \\ & \forall\, i \in \{1 \ldots k-1\}.\ \exists\, G \in \mathcal{T}.\ (n_i, n_{i+1}) \in G \\ U & \text{otherwise.} \end{cases}$$

Recall that $\mathsf{Route}(n_\mathrm{d}, n_1, n_2)$ means in μSAODV that packets addressed to n_d and arriving at n_1 will be forwarded to n_2 as the next hop. The definition then says the network topology should allow for a path from n_1 to n_d via n_2. Using the notion of a consistency mediator, we can define the following property for routing networks.

Definition 4.2 (*Topology consistency*). For network N and network topology $\mathcal{T}$, let $\mu_{\mathcal{T}}$the consistency mediator for N. N is said to be *topology consistent* under attacker M if the equivalence $N \simeq_{\mathcal{T}}^{\mu_{\mathcal{T}}} (N \parallel M)$ holds.

Note that, from the definition of mediated equivalence, M (representing additional, possibly malicious nodes) is allowed to store anything since the convergence predicate is only checked for all $n \in V(N) \cap V(N \parallel M) = V(N)$. However, if interaction of N with M in network $N \parallel M$ causes inconsistent information to be stored by nodes of N, the equivalence in Definition 4.2 cannot be established. Furthermore, the definition does not imply the topology consistency of N, meaning that faults in the protocol are not misinterpreted as attacker actions.

4.3.2. Automated security analysis

Proving the topology consistency condition by hand is error prone for large protocols. We will thus use the analysis framework of Section 3 to establish these results. The following theorem holds:

Theorem 4.3. *For all networks N, M and simple mediators μ, the following implication holds*: *If* $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N \parallel M$ *and* $\forall\ (U, m) \in \sigma.\ \mu(U) = \varepsilon$, *then* $N \simeq_{\mathcal{T}}^{\mu} N \parallel M$.

Proof. By Definition 2.14, we know $N \simeq_{\mathcal{T}}^{\mu} N \parallel M$ iff $N \sqsubseteq_{\mathcal{T}}^{\mu} N \parallel M$ and $N \parallel M \sqsubseteq_{\mathcal{T}}^{\mu} N$. Since μ is simple, i.e. $\mu(U) \in \{\varepsilon, U\}$ for all U, the first inclusion can be proved by Theorem 2.17.

It remains to show $N \parallel M \sqsubseteq_{\mathcal{T}}^{\mu} N$. By definition of $\sqsubseteq_{\mathcal{T}}^{\mu}$, where we note that $V(N) \cap V(N \parallel M) = V(N)$, we have to show the following result:

$$\forall\, U.\, \forall\, n \in V(N).\ N \parallel M \Downarrow_n^{\mathcal{T}} U \Rightarrow N \Downarrow_n^{\mathcal{T}} \mu(U).$$

To prove this by contradiction, we assume its negation:

$$\exists\, U.\, \exists\, n \in V(N).\ N \parallel M \Downarrow_n^{\mathcal{T}} U \wedge \neg(N \Downarrow_n^{\mathcal{T}} \mu(U)).$$

Since $\neg(N \Downarrow_n^{\mathcal{T}} \mu(U))$ it must be that $\mu(U) \neq \varepsilon$ $(*)$, because $N \Downarrow_n^{\mathcal{T}} \varepsilon$ is true for any N. $(*)$ allows us to apply Theorem 3.5 to $N \parallel M \Downarrow_n^{\mathcal{T}} U$ and $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N \parallel M$, both of which we have by assumption, to yield $(U, n) \in \sigma$. However, we know by assumption that $\forall\ (U', m) \in \sigma.\ \mu(U') = \varepsilon$, in particular $\mu(U) = \varepsilon$. This is a contradiction to $(*)$. □

4.3.3. Attackers

It is important to note that—other than in the spi calculus [2], for instance—the algebraic laws of encryption and decryption are not hardcoded into the operational semantics of the calculus. It is thus easily possible (as it would be for example in CSP [28]) to write a process violating perfect cryptography, e.g. to read the contents of an encrypted message without the appropriate key. This is due to the flexibility to define new operations which is offered by CBS$^{\sharp}$ and does not pose a problem, because a correct notion of the attacker can be defined straightforwardly a simple type discipline which is described in the following. Assume the set of function symbols $\mathcal{F}$ is partitioned into the following subsets:

$\mathcal{F}_{\text{oneway}}$	one-way functions
$\mathcal{F}_{\text{twoway}}$	two-way functions
$\mathcal{F}_{\text{secret}}$	secret functions
$\mathcal{F}_{\text{message}}$	message constructors

One- and *two-way functions* signify, as commonly found in cryptographic contexts, functions which are thought to be very difficult to reverse or are easily reversed, respectively. *Secret functions* are introduced for more specific modelling purposes: for example, in Section 2.3 we have introduced functions PrivKey and PubKey to yield key pairs for asymmetric encryption when applied to a seed. These functions are in general not to be applied by any protocol participant and therefore kept secret from the attacker. *Message constructors* are functions which yield messages of valid format within the particular network in question. Conceptually, they are two-way functions, but we decide to introduce them at this point as a simple optimisation to reduce the amount of useless messages created by the attacker.

Example 4.4. Assume that some protocol employs the functions Pair, Hash, Msg, AsymEnc, PubKey, and PrivKey (with their obvious meanings). To specify the attacker properly, an adequate partitioning of this set of functions is given by:

$$\begin{aligned}\textsf{Hash}, \textsf{AsymEnc} &\in \mathcal{F}_{\text{oneway}}\\ \textsf{Pair} &\in \mathcal{F}_{\text{twoway}}\\ \textsf{PubKey}, \textsf{PrivKey} &\in \mathcal{F}_{\text{secret}}\\ \textsf{Msg} &\in \mathcal{F}_{\text{message}}\end{aligned}$$

We can then define an attacker process P_A as follows:

$$\begin{aligned}
&P_1(f) \overset{\text{def}}{=} \textbf{!in}\ f(;x_1,\ldots,x_{\text{arity}(f)}).\,\textbf{store}\ x_1.\ \ldots\textbf{store}\ x_{\text{arity}(f)}\\
&P_2(f) \overset{\text{def}}{=} \textbf{!read}\ x_1.\ \ldots\textbf{read}\ x_{\text{arity}(f)}.\,\textbf{out}\ f(x_1,\ldots,x_{\text{arity}(f)})\\
&P_3(f) \overset{\text{def}}{=} \textbf{!read}\ f(;x_1,\ldots,x_{\text{arity}(f)})\ \textbf{store}\ x_1.\ \ldots\textbf{store}\ x_{\text{arity}(f)}\ \textbf{else nil}\\
&P_4(f) \overset{\text{def}}{=} \textbf{!read}\ x_1.\ \ldots\textbf{read}\ x_{\text{arity}(f)}.\,\textbf{store}\ f(x_1,\ldots,x_{\text{arity}(f)})\\
&P_5 \quad\ \overset{\text{def}}{=} \textbf{!read}\ x_1.\,\textbf{read}\ x_2.\,\textbf{symdec}\ x_1\ x_2\ x\ \textbf{store}\ x\ \textbf{else nil}\\
&\qquad\quad \mid\ \textbf{!read}\ x_1.\,\textbf{read}\ x_2.\,\textbf{asymdec}\ x_1\ x_2\ x\ \textbf{store}\ x\ \textbf{else nil}
\end{aligned}$$

$$P_A \overset{\text{def}}{=} (|_{f\in\mathcal{F}_{\text{message}}}\ P_1(f) \mid P_2(f)) \mid (|_{f\in\mathcal{F}_{\text{twoway}}}\ P_3(f)) \mid (|_{f\in\mathcal{F}_{\text{oneway}}\cup\mathcal{F}_{\text{twoway}}}\ P_4(f)) \mid P_5$$

This follows the Dolev-Yao formalisation in the sense that P_A can

- receive messages and add their components to the store,
- send messages constructed from the store,
- add arguments of 2-way functions to the store,
- apply functions to arguments from the store, and
- perform decryption if keys are in the store.

However, if we add the attacker to a network N to yield $N \parallel n_A[P_A, \varepsilon]$, a major difference to the Dolev-Yao approach is evident: when the network evolves, the attacker is just an ordinary node which has to abide by the topology $\mathcal{T}$. In general, the attacker will neither be able to intercept all messages on the network nor inject messages at all locations. We believe that this reflects accurately the situation in wireless networks: all participants have to abide by purely physical restrictions imposed by radio transmission ranges. In any case, it gives the protocol analyst more freedom, as the classic Dolev-Yao case can be achieved by a careful modelling of $\mathcal{T}$.

4.4. Analysis results for μSAODV

The analysis of μSAODV in a simple scenario shows that the system is indeed not topology consistent. Using the subroutines of Table 8, we can define the following processes and nodes:

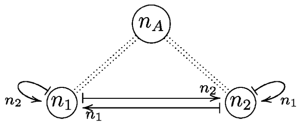
$$\begin{aligned}
&\mathit{MsgHdl}(\mathit{ip}) \overset{\text{def}}{=} \mathit{ReceiveRREQ}(\mathit{ip}) \mid \mathit{ReceiveRREP}(\mathit{ip})\\
&N_1(\mathit{ip}, \mathit{dstip}) \overset{\text{def}}{=} \mathit{ip}[\mathit{SendRREQ}(\mathit{dstip}, \mathit{ip}) \mid \mathit{MsgHdl}(\mathit{ip}), \varepsilon]\\
&N_2(\mathit{ip}) \overset{\text{def}}{=} \mathit{ip}[\mathit{MsgHdl}(\mathit{ip}), \varepsilon]
\end{aligned}$$

The message handler *MsgHdl* represents the main protocol routine. Both N_1 and N_2 are parametrised nodes running this process, however, N_1 is in addition given the capability to initiate a route request. Our scenario then consists of a network of three nodes

$$N_{\text{saodv}} = N_1(n_1, n_2) \parallel N_2(n_2) \parallel n_A[P_A, \varepsilon]$$

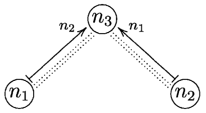
where P_A is defined as in Section 4.3.3 and represents the attack process. Furthermore, a network topology $\mathcal{T}$ is defined such that $E(\mathcal{G}(\mathcal{T})) = \{(n_1, n_A), (n_2, n_A)\}$, thus n_1 and n_2 are never directly connected.

If $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N_{\text{saodv}}$, then Fig. 1 shows the terms contained in σ (computed automatically by our implementation) and their graphical interpretation.



At n_1 : $\mathsf{Route}(n_2, n_1, n_2)$
$\mathsf{Route}(n_2, n_1, n_1)$
At n_2 : $\mathsf{Route}(n_1, n_2, n_2)$
$\mathsf{Route}(n_1, n_2, n_1)$

Fig. 1. Attacker induces topology inconsistency.



At n_1 : $\mathsf{Route}(n_2, n_1, n_3)$
At n_2 : $\mathsf{Route}(n_1, n_2, n_3)$

Fig. 2. Topologically consistent network of honest nodes.

Here, the doubly dotted line represents the abstract network topology, and labelled arrows represent the belief of the nodes about the topology. As the attacker can hide its own name by spoofing sending addresses, the attacker makes n_1 and n_2 believe that they are directly connected, which contradicts the topological situation. Thus, the equivalence of Definition 4.2 does not hold.

As a comparison, Fig. 2 shows the analysis for a network of honest nodes $N_1(n_1, n_2) \parallel N_2(n_2) \parallel N_2(n_3)$ with $E(\mathcal{G}(\mathcal{T})) = \{(n_1, n_3), (n_2, n_3)\}$. The network is topology consistent, as node n_1 correctly believes that there is a route to n_2 via n_3, and this holds analogously for node n_2.

5. Conclusion

In this paper we have presented the broadcast calculus CBS$^\sharp$ and a static analysis to formally analyse secure mobile wireless networks. While at first glance this setting seems to resemble traditional security protocol analysis, we have pointed out that its complications call for new modelling formalisms as well as novel security properties, which are provided and expressible in our framework.

Several directions for future work suggest themselves: for example, the strength of the restrained Dolev-Yao attacker needs more investigation, e.g. under which conditions multiple such attackers are more powerful than a single one and whether a hierarchy of such attackers can be established. Also, the topology consistency property alone does not directly imply what one would understand under “routing security”. The challenge is to find a set of properties implying this goal. Furthermore, it seems there might be a reasonable margin to improve the precision of our static analysis (and then potentially prove more properties), for example, by refining the topology abstraction by using directed and weighted graphs.

References

[1] M. Abadi, C. Fournet, Mobile values, new names, and secure communication, in: Proc. 28th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL’01), ACM Press, New York, 2001, pp. 104–115.

[2] M. Abadi, A.D. Gordon, A calculus for cryptographic protocols: the spi calculus, Inform. and Comput. 148 (1) (1999) 1–70.

[3] L. Bettini, V. Bono, R.D. Nicola, G. Ferrari, D. Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, B. Venneri, The klaim project: theory and practice, in: Global Computing—Programming Environments, Languages, Security and Analysis of Systems, Lecture Notes in Computer Science, Vol. 2874, Springer, Berlin, 2003.

[4] K. Bhargavan, D. Obradovic, C.A. Gunter, Formal verification of standards for distance vector routing protocols, J. ACM 49 (4) (2002) 538–576.

[5] C. Bodei, M. Buchholtz, P. Degano, H.R. Nielson, F. Nielson, Static validation of security protocols, J. Comput. Security 13 (3) (2005) 347–390.

[6] C. Bodei, P. Degano, F. Nielson, H.R. Nielson, Control flow analysis for the pi-calculus, in: Proc. Ninth Internat. Conf. on Concurrency Theory (CONCUR’98), Springer, Berlin, 1998, pp. 84–98.

[7] M. Buchholtz, H.R. Nielson, F. Nielson, A calculus for control flow analysis of security protocols, Internat. J. Inform. Security 2 (3–4) (2004) 145–167.

[8] M. Burrows, M. Abadi, R.M. Needham, A logic of authentication, ACM Trans. Comput. Systems 8 (1) (1990) 18–36.

[9] L. Cardelli, A.D. Gordon, Mobile ambients, in: Proc. First Internat. Conf. on Foundations of Software Science and Computation Structures (FOSSACS'98), Springer, Berlin, 1998.

[10] S. Chiyangwa, M. Kwiatkowska, An analysis of timed properties of AODV, in: Proc. Seventh IFIP Internat. Conf. on Formal Methods for Open Object-based Distributed Systems (FMOODS'05), 2005.

[11] C. Ene, T. Muntean, A broadcast-based calculus for communicating systems, in: Sixth Internat. Workshop on Formal Methods for Parallel Programming: Theory and Applications, 2001.

[12] F.J.T. Fabrega, J. Herzog, J.D. Guttman, Strand spaces: proving security protocols correct, J. Comput. Security (1999) 191–230.

[13] M. Guerrero Zapata, Secure ad hoc on-demand distance vector (SAODV) routing, IETF Internet Draft, 7 February 2006.

[14] M. Guerrero Zapata, N. Asokan, Securing ad hoc routing protocols, in: Proc. 2002 ACM Workshop on Wireless Security (WiSe'02), 2002, pp. 1–10.

[15] Y.-C. Hu, A. Perrig, D.B. Johnson, Ariadne: a secure on-demand routing protocol for ad hoc networks, in: Proc. Eighth ACM Internat. Conf. on Mobile Computing and Networking (MobiCom'02), 2002.

[16] D.B. Johnson, D.A. Maltz, Dynamic source routing in ad hoc wireless networks, in: K. Imielinski (Ed.), Mobile Computing, Vol. 353, Kluwer Academic Publishers, Dordrecht, 1996.

[17] S. Nanz, Specification and security analysis of mobile ad hoc networks, Ph.D. Thesis, Imperial College London, 2006.

[18] S. Nanz, C. Hankin, Static analysis of routing protocols for ad-hoc networks, in: Proc. 2004 ACM SIGPLAN and IFIP WG 1.7 Workshop on Issues in the Theory of Security (WITS'04), 2004, pp. 141–152.

[19] S. Nanz, C. Hankin, Formal security analysis for ad-hoc networks, in: Proc. 2004 Workshop on Views on Designing Complex Architectures (VODCA'04), Electronic Notes in Theoretical Computer Science, Vol. 142, 2006, pp. 195–213.

[20] F. Nielson, H.R. Nielson, C. Hankin, Principles of Program Analysis, Springer, Berlin, 1999.

[21] F. Nielson, H.R. Nielson, H. Sun, M. Buchholtz, R. Rydhof Hansen, H. Pilegaard, H. Seidl, The succinct solver suite, in: Proc. 10th Internat. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04), Lecture Notes in Computer Science, Vol. 2988, Springer, Berlin, 2003, pp. 251–265.

[22] H.R. Nielson, F. Nielson, Flow logic: a multi-paradigmatic approach to static analysis, Essence of Computation: Complexity, Analysis, Transformation (2002) 223–244.

[23] H.R. Nielson, F. Nielson, H. Pilegaard, Spatial analysis of bioambients, in: Static Analysis Symposium (SAS'04), Lecture Notes in Computer Science, Vol. 3148, Springer, Berlin, 2004, pp. 69–83.

[24] L.C. Paulson, The inductive approach to verifying cryptographic protocols, J. Comput. Security 6 (1998) 85–128.

[25] C.E. Perkins, E.M. Belding-Royer, S. Das, Ad hoc on-demand distance vector (AODV) routing, IETF RFC 3561, July 2003.

[26] C.E. Perkins, E.M. Royer, Ad-hoc on demand distance vector routing, in: Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99), 1999.

[27] K.V.S. Prasad, A calculus of broadcasting systems, Sci. Comput. Programming 25 (2–3) (1995) 285–327.

[28] P.Y.A. Ryan, S.A. Schneider, The Modelling and Analysis of Security Protocols: The CSP Approach, Addison-Wesley, Reading, 2001.

[29] K. Sanzgiri, B. Dahill, B.N. Levine, C. Shields, E.M. Belding-Royer, A secure routing protocol for ad hoc networks, in: Proc. 10th IEEE Internat. Conf. on Network Protocols (ICNP'02), 2002.

[30] O. Shivers, Control flow analysis in scheme, in: Proc. ACM SIGPLAN 1988 Conf. on Programming Language Design and Implementation (PLDI'88), ACM Press, New York, 1988, pp. 164–174.

[31] O. Wibling, J. Parrow, A. Pears, Automatized verification of ad hoc routing protocols, in: Proc. 24th IFIP WG 6.1 Internat. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE'04), Lecture Notes in Computer Science, Springer, Berlin, 2004.

[32] I. Zakiuddin, M. Goldsmith, P. Whittaker, P. Gardiner, A methodology for model-checking ad hoc networks, in: Model Checking Software: 10th International SPIN Workshop, Lecture Notes in Computer Science, Vol. 2648, Springer, Berlin, 2003, pp. 181–196.