A Framework for Security Analysis of Mobile Wireless Networks

Sebastian Nanz Chris Hankin

Department of Computing
Imperial College London, UK
{nanz,clh}@doc.ic.ac.uk

17th October 2005

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.

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 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 [10] 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. Indeed we will show in this paper that the protocol is insecure in this respect.

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, and a multitude of effective frameworks have been proposed, e.g. [6, 20, 2, 9] 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 only requested if they are needed), 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 [21] and Dynamic Source Routing (DSR) [13], both on-demand protocols. These are developed for a non-adversarial setting only, and security extensions such as SAODV [11], Ariadne [12], and ARAN [25] 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 [14, 15]). 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 and calculi with security objectives are most closely related to our work. The Calculus of Broadcasting Systems (CBS) [23] 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 [8] 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). 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 allow functions as term constructors and uses them to model cryptography.

Model checking has been used to analyse traditional security protocols specified in a process algebra framework [24]. Static analysis techniques for the same task have been employed by [4] and stimulated our own approach. [3, 28, 7, 27] 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 [11, 12, 25], who 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 CBS# 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 CBS#, which can be used to automatically derive the topology consistency condition.

1.4 Overview of the Paper

In Section 2, the syntax and operational semantics of the calculus CBS# 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 SAODV-basic, a simplified version of SAODV, as a worked example of protocol modelling in CBS#. We formalise topology consistency as an important property in the setting of routing protocols and apply our analysis to show that SAODV-basic violates this property. We conclude with notes on future work in Section 5.

2 The Calculus CBS#

In this section we present CBS#, a process calculus for modelling mobile wireless networks. It inherits broadcast as the base communication paradigm from the Calculus of Broadcasting Systems (CBS) [23], 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 CBS# 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# 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 $arity : \mathcal{F} \to \mathbb{N}$ to yield the arity of a function symbol. The set of terms T consists of names $n \in \mathcal{N}$, variables $x \in \mathcal{X}$, and function applications $\mathsf{F}(T_1, \ldots, T_k)$ where $T_i \in \mathsf{T}$, $\mathsf{F} \in \mathcal{F}$ and $arity(\mathsf{F}) = k$.

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 $[U_1/x_1] \ldots [U_k/x_k]$. Free names can be substituted by other names analogously.

Processes. The set of processes P is inductively defined as follows: The terminated process is represented by nil. The sending of a ground term T is denoted by $\mathsf{out}\ T.P$, 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 $\mathsf{in}\ x.P$ expresses readiness to receive a ground term T and bind it to x.

The action $\mathsf{store}\ T.P$ denotes storage of a ground term T in a private store S introduced below. Terms are retrieved from the store by the action $\mathsf{read}\ x.P$ which binds a random term $T \in S$ to x.

Terms

T	$::=$	n	names, $n \in \mathcal{N}$
	$\mid$	x	variables, $x \in \mathcal{X}$
	$\mid$	$\mathsf{F}(\widetilde{T})$	function application, $\mathsf{F} \in \mathcal{F}$

Processes

P	$::=$	nil	termination
	$\mid$	$\mathsf{out}\ T.P$	sending
	$\mid$	$\mathsf{in}\ x.P$	reception
	$\mid$	$\mathsf{store}\ T.P$	storage
	$\mid$	$\mathsf{read}\ x.P$	retrieval
	$\mid$	$\mathsf{case}\ T\ \mathsf{of}\ \mathsf{F}(\widetilde{T}; \widetilde{x})\ P_1\ \mathsf{else}\ P_2$	case distinction, $\mathsf{F} \in \mathcal{F}$
	$\mid$	$A(\widetilde{T})$	constant
	$\mid$	$P_1 \mid P_2$	parallel composition

Stores

S	$::=$	$\{T\}$	singleton
	$\mid$	$S_1 \cup S_2$	union

Networks

N	$::=$	$n[P, S]$	node, $n \in \mathcal{N}_{loc} \subseteq \mathcal{N}$
	$\mid$	$N_1 \parallel N_2$	parallel composition

Table 1: Syntax of CBS#

The action for case distinction $\mathsf{case}\ T\ \mathsf{of}\ \mathsf{F}(\widetilde{T}; \widetilde{x})\ P_1\ \mathsf{else}\ P_2$ tries to match a term T with the term $\mathsf{F}(\widetilde{T}; \widetilde{x})$ and continues on success with P_1 in which variables $\widetilde{x}$ are bound, or otherwise with P_2. In order to match, T has to be of the form $\mathsf{F}(\widetilde{T}, \widetilde{U})$ with $|\widetilde{U}| = |\widetilde{x}|$. Constants can be defined as $A(\widetilde{U}) \stackrel{\text{def}}{=} P$, where $fv(P) \subseteq \widetilde{U}$. Definitions are then invoked by $A(\widetilde{T})$. Processes can be executed in parallel, written as $P_1 \mid P_2$. Multiple parallel actions are abbreviated by $|_{i \in \mathcal{I}}\ P_i$.

Stores. Stores are non-empty sets of terms. The usual set operations such as element $\in$ are defined for them. They are defined non-empty to prevent the read operation from getting stuck. An initialisation of stores which is always suitable can be achieved by requiring $\varepsilon \in S$, where ε is the empty function which matches with no other term. Unless otherwise specified, we will assume this initialisation.

Networks. Networks $N \in \mathsf{N}$ 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 $G = (V, E)$ of sets satisfying $E \subseteq V \times V$. V is called the set of vertices and is referred to as $V(G)$, and E or $E(G)$ is the set of edges. We do not allow self-loops, i.e. $(n, n') \in E(G)$ implies $n \neq n'$.

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 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
$N \rightarrow N'$	Reduction relation
$N \equiv N'$	Structural equivalence

Labelled transition relation. $N \xrightarrow{(U,m)\sharp}_G N'$ describes the evolution of a network N to a network N' during transmission of a term and abiding by the connectivity graph G. The label $(U, m)\sharp$ shows the transmitted (variable-free) term U and the location name $m \in \mathcal{N}_{loc}$ of its sender, as well as a mode identifier $\sharp \in \{!, ?, :\}$. The pair (U, m) is called the message. The different modes denote sending $(U, m)!$, receiving $(U, m)?$, and losing (U, m): a message. Mode identifiers $\sharp_1$ and $\sharp_2$ can be composed according to the following algebra:

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

Here, $\perp$ expresses that two sending actions cannot be combined within one derivation.

The relation $\xrightarrow{(U,m)\sharp}_G$ 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.

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

Definition 2.1 ($\boldsymbol{\mathcal{T}}$-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. The reflexive, transitive closure of $\mathcal{T}$-faithful relations is denoted $\longrightarrow^*_{\mathcal{T}}$.

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

Rule Nil expresses that the node running the nil process $n[\mathsf{nil}, S]$ loses any message (U, m). Rules Out$_1$ and Out$_2$ describe the behaviour of a sender $n[\mathsf{out}\ T.P, S]$. 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 $n[\mathsf{in}\ x.P, S]$ 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 gets bound to U so that the continuation process is $P[U/x]$. If however $(m, n) \notin E(G)$, any message will be discarded by rule In$_2$.

The rule for parallelism PPar makes use of the algebra for the composition $\circ$ of $\sharp_1, \sharp_2 \in \{!, ?, :\}$ as defined above. 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 discarding a particular message. It is ensured that *at most one* process is sending by requiring $\sharp_1 \circ \sharp_2 \neq \perp$ (note $! \circ ! = \perp$) and by the property "$\sharp_1 \circ \sharp_2 \in \{?, :\} \Rightarrow \sharp_1, \sharp_2 \neq !$" of the algebra.

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

Table 2: Labelled transition relation

Example 2.2 Let three nodes be defined as follows

$$\begin{aligned} N_1 &= n_1[\mathsf{out}\ T.\mathsf{nil}, S_1] \\ N_2 &= n_2[\mathsf{in}\ x.\mathsf{nil}, S_2] \\ N_3 &= n_3[\mathsf{in}\ x.\mathsf{nil}, S_3]. \end{aligned}$$

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

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

If furthermore $E(G_2) = \{(n_1, n_3)\}$ and $\mathcal{T} = \{G_1, G_2\}$ then we can conclude that the final configuration will either be $n_1[\mathsf{nil}, S_1] \parallel n_2[\mathsf{in}\ x.\mathsf{nil}, S_2] \parallel n_3[\mathsf{nil}, S_3]$ or $n_1[\mathsf{nil}, S_1] \parallel n_2[\mathsf{nil}, S_2] \parallel n_3[\mathsf{in}\ x.\mathsf{nil}, S_3]$.

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

In rule Store, $n[\mathsf{store}\ T.P, S]$ adds a term T to the store S. Rule Read for retrieval $n[\mathsf{read}\ x.P, S]$ binds a random term $T \in S$ to x in P. In the rules for case distinction,

$$\textsc{Store} \qquad n[\mathsf{store}\ T.P, S] \rightarrow n[P, S \cup \{T\}]$$

$$\textsc{Read} \qquad \frac{T \in S}{n[\mathsf{read}\ x.P, S] \rightarrow n[P[T/x], S]}$$

$$\textsc{Case}_1 \qquad \frac{T = \mathit{F}(\widetilde{T}, \widetilde{U}) \qquad |\widetilde{U}| = |\widetilde{x}|}{n[\mathsf{case}\ T\ \mathsf{of}\ \mathit{F}(\widetilde{T}; \widetilde{x})\ P_1\ \mathsf{else}\ P_2, S] \rightarrow n[P_1[\widetilde{U}/\widetilde{x}], S]}$$

$$\textsc{Case}_2 \qquad \frac{\nexists \widetilde{U}.\, T = \mathit{F}(\widetilde{T}, \widetilde{U}) \wedge |\widetilde{U}| = |\widetilde{x}|}{n[\mathsf{case}\ T\ \mathsf{of}\ \mathit{F}(\widetilde{T}; \widetilde{x})\ P_1\ \mathsf{else}\ P_2, S] \rightarrow n[P_2, S]}$$

$$\textsc{Const} \qquad \frac{A(\widetilde{U}) \overset{\text{def}}{=} P \qquad n[P[\widetilde{T}/\widetilde{U}], S] \rightarrow n[P', S']}{n[A(\widetilde{T}), S] \rightarrow n[P', S']}$$

Table 3: Reduction relation

the term T is matched against a template $\mathit{F}(\widetilde{T}; \widetilde{x})$. In rule $\textsc{Case}_1$, if T is of the form $\mathit{F}(\widetilde{T}, \widetilde{U})$ and $|\widetilde{U}| = |\widetilde{x}|$, the continuation binds the terms $\widetilde{U}$ against the variables $\widetilde{x}$ to yield $n[P_1[\widetilde{U}/\widetilde{x}], S]$. Otherwise, rule $\textsc{Case}_2$ demands P_2 as continuation process. Constants $n[A(\widetilde{T}), S]$ get expanded whenever necessary as shown in rule $\textsc{Const}$. We assume though that possible recursive appearances $A(\widetilde{T}')$ imply $\widetilde{T} = \widetilde{T}'$. This means that we limit infinite behaviour to *replication* which is important for the safety of our analysis in Section 3.

Example 2.3 Assume the following definition:

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

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

$$N \rightarrow n[\mathsf{store}\ \mathit{Msg}.\mathsf{nil}, S] \rightarrow n[\mathsf{nil}, S \cup \{\mathit{Msg}\}]$$

For $T = \mathit{Env}(m; \mathit{Msg})$ we would however get $n[\mathsf{nil}, S]$ 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 parallel processes are equivalent to parallel nodes running these processes, and stores can be partitioned arbitrarily.

Example 2.4 We have the following equivalences, where the second holds because of Red and Store, the other two because of Par.

$$n[\mathsf{store}\ T.\mathsf{nil} \mid \mathsf{in}\ x.\mathsf{nil}, S] \equiv n[\mathsf{store}\ T.\mathsf{nil}, S] \parallel n[\mathsf{in}\ x.\mathsf{nil}, S] \equiv$$
$$\equiv n[\mathsf{nil}, S \cup \{T\}] \parallel n[\mathsf{in}\ x.\mathsf{nil}, S] \equiv n[\mathsf{nil} \mid \mathsf{in}\ x.\mathsf{nil}, S \cup \{T\}]$$

2.3 Notational Conventions and Cryptographic Primitives

For the specification of even moderately large protocols such as SAODV-basic 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 line breaks and indentation as exemplified in Table 8, and we omit a trailing nil whenever no confusion arises. In this section we show some more notational conventions and show a way to model cryptographic primitives in CBS#.

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.

$$\begin{aligned}&\mathsf{if}\ T = U\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \equiv\\ &\quad \mathsf{case}\ \mathit{Match}(T)\ \mathsf{of}\ \mathit{Match}(U;\,)\ P_1\ \mathsf{else}\ P_2\end{aligned}$$

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

$$\begin{aligned}&\mathsf{in}\ F(\widetilde{T};\widetilde{x}).P \equiv\\ &\quad \mathsf{in}\ x.\mathsf{case}\ x\ \mathsf{of}\ F(\widetilde{T};\widetilde{x})\ P\ \mathsf{else\ nil}\end{aligned}$$

$$\begin{aligned}&\mathsf{read}\ F(\widetilde{T};\widetilde{x})\ P_1\ \mathsf{else}\ P_2 \equiv\\ &\quad \mathsf{read}\ x.\mathsf{case}\ x\ \mathsf{of}\ F(\widetilde{T};\widetilde{x})\ P_1\ \mathsf{else}\ P_2\end{aligned}$$

Cryptographic Primitives. In order to express public-key digital signatures we use key pairs $(\mathit{PubKey}(seed), \mathit{PrivKey}(seed))$ created from the same *seed* by applying functions *PubKey* and *PrivKey*. A signature of term T under private key $\mathit{PrivKey}(seed)$ then simply corresponds to applying the function *Sign* to yield $\mathit{Sign}(\mathit{PrivKey}(seed), T)$. 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,

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	$\dfrac{N \equiv N'' \quad N'' \equiv N'}{N \equiv N'}$
Compose	$\dfrac{N_1 \equiv N_1'}{N_1 \parallel N_2 \equiv N_1' \parallel N_2}$
Red	$\dfrac{N \rightarrow N'}{N \equiv N'}$
Par	$n[P_1 \mid P_2, S_1 \cup S_2] \equiv n[P_1, S_1] \parallel n[P_2, S_2]$

Table 4: Structural equivalence

$seed \notin fv(P_1) \cup fv(P_2)$.

```
checksig $sig$ $pubkey$ $T$ $P_1$ else $P_2 \equiv$
    case $pubkey$ of $\mathit{PubKey}(;seed)$
        case $sig$ of $\mathit{Sign}(\mathit{PrivKey}(seed), T;)$ $P_1$ 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 $msg$ $privkey$ $x$ $P_1$ else $P_2 \equiv$
    case $privkey$ of $\mathit{PrivKey}(;seed)$
        case $msg$ of $\mathit{AsymEnc}(\mathit{PubKey}(seed); x)$ $P_1$ else $P_2$
    else
        $P_2$
```

```
symdec $msg$ $\mathit{SymKey}(seed)$ $x$ $P_1$ else $P_2 \equiv$
    case $msg$ of $\mathit{SymEnc}(\mathit{SymKey}(seed); x)$ $P_1$ else $P_2$
```

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

2.4 Behavioural Equivalences

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

Definition 2.5 ($\mathcal{T}$-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.

Example 2.6 The following propositions are proved by defining a set $\mathcal{S} \subseteq \mathsf{N} \times \mathsf{N}$ and proving that it is a $\mathcal{T}$-bisimulation.

(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,

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

(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 2.7 (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 2.8

(1) $n[\mathsf{nil}, \{\varepsilon\}] \parallel N \sim N$

(2) If $n \notin V(N)$ then $n[\mathsf{nil}, S] \parallel N \sim N$.

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 5: Barb predicate

(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) $n[\mathsf{nil}, S] \sim n'[\mathsf{nil}, S']$

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 transmits $(T, n)!$, a node with the same name n in the corresponding network is also ready to transmit $(T, n)!$.
- Whenever one or more nodes receive $(U, m)?$, one or more nodes in the corresponding network are also ready to receive $(U, m)?$.
- Whenever the complete network refuses a message (U, m):, all the nodes of the corresponding network refuse 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 2.8 (3) and (4). However, $n[\mathsf{store}\ T.P, S] \nsim n[P, S]$ 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.

Definition 2.9 (Barbed Equivalence) A $\mathcal{T}_{max}$-simulation $\mathcal{S}$ is called a *barbed simulation* if $N\,\mathcal{S}\,M$ implies for each barb U and all $n \in V(N) \cap V(M)$: if $N \downarrow_n U$ then $M \downarrow_n U$.

Conv-Barb	$\dfrac{N \downarrow_n U}{N \Downarrow_n^{\mathcal{T}} U}$
Conv-Comm	$\dfrac{N \xrightarrow{m!}_{G \in \mathcal{T}} N' \qquad N' \Downarrow_n^{\mathcal{T}} U}{N \Downarrow_n^{\mathcal{T}} U}$

Table 6: Convergence predicate

$\mathcal{S}$ is called a barbed bisimulation if both $\mathcal{S}$ and its converse are barbed simulations. *Barbed equivalence*, written $\stackrel{\bullet}{\sim}$, is the largest barbed bisimulation.

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

Example 2.10 $n[\mathsf{store}\ T.\mathsf{nil}, S] \not\stackrel{\bullet}{\sim} n[\mathsf{nil}, S']$

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

Theorem 2.11

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

(2) $N \stackrel{\bullet}{\sim} M$ 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 and holds if N will eventually store U (possibly after some interactions under network topology $\mathcal{T}$) at location n.

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

Definition 2.12 (Mediated Equivalence) For given topology $\mathcal{T}$ and function $c : \mathsf{T} \to \mathsf{T}$, called a *mediator*, we write $N \sqsubseteq_{\mathcal{T}}^{c} M$ if for any term U and for all $n \in V(N) \cap V(M)$: $N \Downarrow_n^{\mathcal{T}} U$ implies $M \Downarrow_n^{\mathcal{T}} c(U)$. *Mediated equivalence*, written $\simeq_{\mathcal{T}}^{c}$, is then defined as: $N \simeq_{\mathcal{T}}^{c} M$ iff $N \sqsubseteq_{\mathcal{T}}^{c} M$ and $M \sqsubseteq_{\mathcal{T}}^{c} N$.

The mediator $c : \mathsf{T} \to \mathsf{T}$ 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 2.13 Let a mediator c_S and two networks N and $M(U)$ be defined as follows.

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

Then, the following holds because $N \Downarrow_n^{\mathcal{T}} c_S(T_{sec})$:

$$N \parallel M(T_{sec}) \simeq_{\mathcal{T}}^{c_S} N$$

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 $\sqsubseteq_{\mathcal{T}}^{c}$ and rule PPar:

Theorem 2.14 *For all networks N, M and mediators c with $c(U) \in \{\varepsilon, U\}$ for all U, the following holds:*

$$N \sqsubseteq_{\mathcal{T}}^{c} N \parallel M$$

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

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

If $c(U) = \varepsilon$, we can show $N \parallel M \Downarrow_n^{\mathcal{T}} c(U)$ to hold directly with Conv-Barb and Barb-Empty.

Fix thus $U \neq \varepsilon$ and $n \in V(N)$, and assume $N \Downarrow_n^{\mathcal{T}} U$. 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 $M'' \xrightarrow{(U,m)\natural}_{G \in \mathcal{T}} M'$ for either $\natural = ?$ or $\natural = :$. Using PPar on $N'' \xrightarrow{(U,m)!}_{G \in \mathcal{T}} N'$ and $M'' \xrightarrow{(U,m)\natural}_{G \in \mathcal{T}} M'$ 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 $N' \parallel M' \Downarrow^{\mathcal{T}}_n U$. By finitely many applications of Conv-Comm on $(**)$ we have $N \parallel M \Downarrow^{\mathcal{T}}_n U$. □

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 [26], it has since then been applied to a variety of programming paradigms, including calculi for concurrency and security [5, 4].

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 (Section 4) to automatically check whether networks are mediated equivalent and prove a security property for network protocols specified in CBS#.

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 thus be safe by defining a static abstraction $\mathcal{G}(\mathcal{T})$ for $\mathcal{T}$ in the following way:

$$\mathcal{G}(\mathcal{T}) = (\textstyle\bigcup_{G\in\mathcal{T}} V(G), \bigcup_{G\in\mathcal{T}} E(G))$$

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 $\textsc{In}_1$ 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 $\textsc{In}_2$ because $\textsc{In}_2$ does not lead to the execution of an action.

3.2 Specification

We specify the analysis by a *flow logic* [19, 17], 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:

$$\begin{array}{ll} (\kappa, \sigma) \models^{\mathcal{C},\theta}_{\mathcal{G}(\mathcal{T}),n} P & \text{judgement for processes} \\ (\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N & \text{judgement for networks} \end{array}$$

Both judgements are parameterised with a *recursion environment* $\mathcal{C}$, which is used to ensure termination of the analysis in presence of recursive appearances of constants. The rules ensure that $(A(\widetilde{T}), n) \in \mathcal{C}$ for $A(\widetilde{U}) \stackrel{\text{def}}{=} P$ whenever P is being analysed, and recursive appearances of $A(\widetilde{T})$ are replaced by nil.

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

$$\begin{array}{lcll} \kappa & \subseteq & \mathsf{T} \times \mathcal{N}_{loc} & \text{network cache} \\ \sigma & \subseteq & \mathsf{T} \times \mathcal{N}_{loc} & \text{store cache} \end{array}$$

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

$$\theta \ \subseteq\ \mathsf{T} \times \mathcal{X} \quad \text{substitution environment}$$

Judgement for Processes

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

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

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

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

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

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

CFA-Const $(\kappa,\sigma) \models^{\mathcal{C},\theta}_{\mathcal{G}(\mathcal{T}),n} A(\widetilde{T})$

$$\text{iff} \quad \begin{cases} (\kappa,\sigma) \models^{\mathcal{C},\theta}_{\mathcal{G}(\mathcal{T}),n} \mathsf{nil} & \text{if } (A(\widetilde{T}),n) \in \mathcal{C} \\ (\kappa,\sigma) \models^{\mathcal{C}\cup\{(A(\widetilde{T}),n)\},\theta}_{\mathcal{G}(\mathcal{T}),n} P[\widetilde{T}/\widetilde{U}] & \text{if } (A(\widetilde{T}),n) \notin \mathcal{C} \text{ and } A(\widetilde{U}) \overset{\text{def}}{=} P \end{cases}$$

CFA-Par $(\kappa,\sigma) \models^{\mathcal{C}_1\cup\mathcal{C}_2,\theta}_{\mathcal{G}(\mathcal{T}),n} P_1 \mid P_2$
iff $(\kappa,\sigma) \models^{\mathcal{C}_1,\theta}_{\mathcal{G}(\mathcal{T}),n} P_1 \wedge (\kappa,\sigma) \models^{\mathcal{C}_2,\theta}_{\mathcal{G}(\mathcal{T}),n} P_2$

Judgement for Networks

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

CFA-PPar $(\kappa,\sigma) \models^{\mathcal{C}_1\cup\mathcal{C}_2}_{\mathcal{G}(\mathcal{T})} N_1 \parallel N_2$
iff $(\kappa,\sigma) \models^{\mathcal{C}_1}_{\mathcal{G}(\mathcal{T})} N_1 \wedge (\kappa,\sigma) \models^{\mathcal{C}_2}_{\mathcal{G}(\mathcal{T})} N_2$

Table 7: Control Flow Analysis for CBS#

Again intuitively, if $(U,x) \in \theta$, written $\theta[U/x]$, the term U may be bound to variable x during execution of P. The empty substitution environment is denoted $\varnothing$. The use of a local environment allows for a more precise analysis of the case statement in our language which in turn proves to be crucial for limiting the number of false positives arising from the security analysis.

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 $\mathsf{out}\ T.P$ 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, $(U, n) \in \sigma$. On the other hand, rule CFA-Read resembles rule CFA-In: terms are now 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 $\mathsf{F}(V_1 \ldots V_k)$, it is checked that $T_1\theta = V_1 \wedge \ldots \wedge T_j\theta = V_j$, 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 $i = j+1, \ldots, k$, meaning the remaining arguments of F are bound to the variables x_i.

Rule CFA-Const says that constants are expanded and analysed once, possible recursive appearances in the continuation (then contained in $\mathcal{C}$) are replaced by nil. Rule CFA-Par is 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. As flow logic is a semantics based approach, we prove the analysis estimate correct with respect to the operational semantics of Section 2.2. Well-definedness and Moore family result for the analysis are straightforward using the techniques described in [17] and therefore not included in this report. The main result is a subject reduction theorem which is proved in this section and states 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 *For all* $(A(\widetilde{T}), n) \in \mathcal{C}$ *let* $(\kappa, \sigma) \vDash^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} P[\widetilde{T}/\widetilde{U}]$ *if* $A(\widetilde{U}) \overset{\text{def}}{=} P$*. Then the following implications hold:*

(1) if $N \to N'$ *and* $(\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N$*, then* $(\kappa, \sigma) \vDash^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} N'$*, and*

(2) if $N \equiv N'$ *and* $(\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N$*, then* $(\kappa, \sigma) \vDash^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} N'$*,*

where $\mathcal{C} \subseteq \mathcal{C}'$ *and for all* $(A(\widetilde{T}), n) \in \mathcal{C}'\backslash\mathcal{C}$ *we have* $(\kappa, \sigma) \vDash^{\mathcal{C}',\varnothing}_{\mathcal{G}(\mathcal{T}),n} P[\widetilde{T}/\widetilde{U}]$ *if* $A(\widetilde{U}) \overset{\text{def}}{=} P$*.*

Proof. (1) The proof is by induction on the shape of the derivation tree for $N \to N'$.

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

$$\begin{array}{lll}
 & (\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[\mathsf{store}\ T.P, S] & \text{(by assumption)} \\
\text{thus} & (\kappa, \sigma) \vDash^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} \mathsf{store}\ T.P \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Node)} \\
\text{thus} & (T, n) \in \sigma \wedge (\kappa, \sigma) \vDash^{\mathcal{C}',\varnothing}_{\mathcal{G}(\mathcal{T}),n} P \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Store)} \\
\text{thus} & (\kappa, \sigma) \vDash^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} P \wedge \forall\, U \in S \cup \{T\}.\ (U, n) \in \sigma & \\
\text{thus} & (\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[P, S \cup \{T\}] & \text{(by CFA-Node)}
\end{array}$$

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

$$\begin{array}{lll}
 & (\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[\mathsf{read}\ x.P, S] & \text{(by assumption)} \\
\text{thus} & (\kappa, \sigma) \vDash^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} \mathsf{read}\ x.P \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Node)} \\
\text{thus} & \forall\, (U, n) \in \sigma.\ (\kappa, \sigma) \vDash^{\mathcal{C},[U/x]}_{\mathcal{G}(\mathcal{T}),n} P \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Read)} \\
\text{thus} & (\kappa, \sigma) \vDash^{\mathcal{C},[T/x]}_{\mathcal{G}(\mathcal{T}),n} P \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by } (*)\text{)} \\
\text{thus} & (\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[P[T/x], S] & \text{(by CFA-Node)}
\end{array}$$

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

$$\begin{array}{lll}
 & (\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[\mathsf{case}\ T\ \mathsf{of}\ \mathsf{F}(T_1 \dots T_j; x_{j+1} \dots x_k)\ P_1\ \mathsf{else}\ P_2, S] & \text{(by assumption)} \\
\text{thus} & (\kappa, \sigma) \vDash^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} \mathsf{case}\ T\ \mathsf{of}\ \mathsf{F}(T_1 \dots T_j; x_{j+1} \dots x_k)\ P_1\ \mathsf{else}\ P_2 \wedge & \\
 & \quad \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Node)} \\
\text{thus} & (T = \mathsf{F}(V_1 \dots V_k) \wedge \bigwedge_{i=1}^{j} T_i = V_i \Rightarrow (\kappa, \sigma) \vDash^{\mathcal{C},[V_i/x_i]_{i=j+1}^{k}}_{\mathcal{G}(\mathcal{T}),n} P_1) \wedge & \\
 & \quad \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Case)} \\
\text{thus} & (\kappa, \sigma) \vDash^{\mathcal{C},[U_i/x_i]_{i=j+1}^{k}}_{\mathcal{G}(\mathcal{T}),n} P_1 \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by } (*)\text{)} \\
\text{thus} & (\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[P_1[\widetilde{U}/\widetilde{x}], S] & \text{(by CFA-Node)}
\end{array}$$

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

$$\begin{array}{lll}
 & (\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[\mathsf{case}\ T\ \mathsf{of}\ \mathsf{F}(\widetilde{T};\widetilde{x})\ P_1\ \mathsf{else}\ P_2, S] & \text{(by assumption)}\\
\text{thus} & (\kappa,\sigma) \models^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} \mathsf{case}\ T\ \mathsf{of}\ \mathsf{F}(\widetilde{T};\widetilde{x})\ P_1\ \mathsf{else}\ P_2 \wedge \forall\, U \in S.\ (U,n) \in \sigma & \text{(by CFA-Node)}\\
\text{thus} & (\kappa,\sigma) \models^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} P_2 \wedge \forall\, U \in S.\ (U,n) \in \sigma & \text{(by CFA-Case)}\\
\text{thus} & (\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[P_2, S] & \text{(by CFA-Node)}
\end{array}$$

Case Const. Then $N = n[A(\widetilde{T}), S]$ and $N' = n[P', S']$.

$$\begin{array}{lll}
 & (\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[A(\widetilde{T}), S] & \text{(by assumption)}\\
\text{thus} & (\kappa,\sigma) \models^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} A(\widetilde{T}) \wedge \forall\, U \in S.\ (U,n) \in \sigma & \text{(by CFA-Node)}
\end{array}$$

We have to distinguish two subcases when applying CFA-Const:

(a) Assume $(A(\widetilde{T}), n) \in \mathcal{C}$. From the preconditions of Const we have $A(\widetilde{U}) \stackrel{\text{def}}{=} P$ and thus $(\kappa,\sigma) \models^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} P[\widetilde{T}/\widetilde{U}]$ from the preconditions of the lemma. With CFA-Node we have $(\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[P[\widetilde{T}/\widetilde{U}], S]$. Furthermore, we know $n[P[\widetilde{T}/\widetilde{U}], S] \to n[P', S']$ from Const. We can thus apply the induction hypothesis to have $(\kappa,\sigma) \models^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} n[P', S']$, and the postconditions of the theorem hold for $\mathcal{C}'$ because they hold for the induction hypothesis.

(b) Assume $(A(\widetilde{T}), n) \notin \mathcal{C}$. Then we have $(\kappa,\sigma) \models^{\mathcal{C}\cup\{(A(\widetilde{T}),n)\},\varnothing}_{\mathcal{G}(\mathcal{T}),n} P[\widetilde{T}/\widetilde{U}]$ from CFA-Const. By taking $\mathcal{C}' = \mathcal{C} \cup \{(A(\widetilde{T}), n)\}$, the postconditions of the lemma are fulfilled. Furthermore, we can apply the induction hypothesis as in (a) to have $(\kappa,\sigma) \models^{\mathcal{C}''}_{\mathcal{G}(\mathcal{T})} n[P', S']$.

(2) The proof is by induction on the shape of the derivation tree for $N \equiv N'$.

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

$$\begin{array}{lll}
 & (\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_1 \parallel N_2 & \text{(by assumption)}\\
\text{thus} & (\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_1 \wedge (\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_2 & \text{(by CFA-PPar)}\\
\text{thus} & (\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_2 \parallel N_1 & \text{(by CFA-PPar)}
\end{array}$$

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 $(\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N$ by assumption. The induction hypothesis applied to $N' \equiv N$ is

$$N' \equiv N \wedge (\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N' \Rightarrow (\kappa,\sigma) \models^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} N$$

Since $\mathcal{C} \subseteq \mathcal{C}'$, $(\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N$, and $N' \equiv N$ are true, $(\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N'$ must be true as well.

Case Trans. We have $(\kappa,\sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N$ by assumption and can apply the induction hypothesis first to $N \equiv N''$ to get $(\kappa,\sigma) \models^{\mathcal{C}''}_{\mathcal{G}(\mathcal{T})} N''$, and thus a second time to $N'' \equiv N'$ to get $(\kappa,\sigma) \models^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} N'$.

Case COMPOSE. Then $N = N_1 \parallel N_2$ and $N' = N'_1 \parallel N_2$. We have $(\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_1 \parallel N_2$ and with CFA-PPAR also $(\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_1$ and $(\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_2$. We can thus apply the induction hypothesis to $N_1 \equiv N'_1$ to get $(\kappa, \sigma) \models^{\mathcal{C}'_1}_{\mathcal{G}(\mathcal{T})} N'_1$. Together with $(\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_2$ we have $(\kappa, \sigma) \models^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} N'_1 \parallel N_2$ as desired with CFA-PPAR.

Case RED. We have $(\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N$ by assumption and $N \to N'$ by RED and can apply Lemma 3.1 (1) to yield $(\kappa, \sigma) \models^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} N'$.

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

$$\begin{array}{lll}
 & (\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[P_1 \mid P_2, S_1 \cup S_2] & \text{(by assumption)} \\
\text{thus} & (\kappa, \sigma) \models^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} P_1 \mid P_2 \wedge \forall\, U \in S_1 \cup S_2.\ (U, n) \in \sigma & \text{(by CFA-NODE)} \\
\text{thus} & (\kappa, \sigma) \models^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} P_1 \wedge (\kappa, \sigma) \models^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} P_2 \wedge & \\
 & \quad \forall\, U \in S_1.\ (U, n) \in \sigma \wedge \forall\, U \in S_2.\ (U, n) \in \sigma & \text{(by CFA-PAR)} \\
 & (\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[P_1, S_1] \parallel n[P_2, S_2] & \text{(by CFA-NODE)}
\end{array}$$
□

The following lemma simplifies the proofs of Theorem 3.3 and Theorem 3.4.

Lemma 3.2 *For all $(A(\widetilde{T}), n) \in \mathcal{C}$ let $(\kappa, \sigma) \models^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} P[\widetilde{T}/\widetilde{U}]$ if $A(\widetilde{U}) \overset{\text{def}}{=} P$. Then the following implication holds:*

$$\textit{if } N \xrightarrow{(T,n)!}_{G \in \mathcal{T}} N' \textit{ and } (\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N, \textit{ then } (T, n) \in \kappa.$$

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

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

Case PPAR. Then $N = N_1 \parallel N_2$. We have $(\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_1 \parallel N_2$ by assumption, and with CFA-PPAR thus $(\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_1$ and $(\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_2$. 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 $(\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_1$ or $(\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N_2$, whichever is relevant, the induction hypothesis can be applied to give $(T, n) \in \kappa$.

Case STRUCT. We have $(\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N$ 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 $(\kappa, \sigma) \models^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N$ gives $(\kappa, \sigma) \models^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} M$, and the preconditions of the induction hypothesis are fulfilled. Thus, the induction hypothesis can be applied to give $(T, n) \in \kappa$. □

Using Lemma 3.1 and 3.2, we have the following 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) *For all* $(A(\widetilde{T}), n) \in \mathcal{C}$ *let* $(\kappa, \sigma) \vDash^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} P[\widetilde{T}/\widetilde{U}]$ *if* $A(\widetilde{U}) \overset{\text{def}}{=} P$. *Then the following implication holds:*

$$\textit{if } N \xrightarrow{(U,m)!}_{G \in \mathcal{T}} N' \textit{ and } (\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N, \textit{ then } (\kappa, \sigma) \vDash^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} N',$$

where $\mathcal{C} \subseteq \mathcal{C}'$ *and for all* $(A(\widetilde{T}), n) \in \mathcal{C}' \backslash \mathcal{C}$ *we have* $(\kappa, \sigma) \vDash^{\mathcal{C}',\varnothing}_{\mathcal{G}(\mathcal{T}),n} P[\widetilde{T}/\widetilde{U}]$ *if* $A(\widetilde{U}) \overset{\text{def}}{=} P$.

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

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

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

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

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

Case In$_1$**.** Then $M = n[\mathsf{in}\, x.P, S]$ and $M' = n[P[U/x], S]$. By assuming the preconditions of the theorem, $N \xrightarrow{(U,m)!}_{G \in \mathcal{T}} N'$ and $(\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N$, Lemma 3.2 gives $(U, m) \in \kappa$ $(*)$. From the preconditions of In$_1$ we know $(m, n) \in E(G)$, and with Equation (1) on page 18 we have $(m, n) \in E(\mathcal{G}(\mathcal{T}))$ $(**)$.

$$\begin{array}{lll}
 & (\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[\mathsf{in}\, x.P, S] & \text{(by assumption)} \\
\text{thus} & (\kappa, \sigma) \vDash^{\mathcal{C},\varnothing}_{\mathcal{G}(\mathcal{T}),n} \mathsf{in}\, x.P \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-Node)} \\
\text{thus} & (\forall\, (U', m') \in \kappa.\ (m', n) \in E(\mathcal{G}(\mathcal{T})) \Rightarrow (\kappa, \sigma) \vDash^{\mathcal{C},[U'/x]}_{\mathcal{G}(\mathcal{T}),n} P) \wedge & \\
 & \quad \forall\, U \in S.\ (U, n) \in \sigma & \text{(by CFA-In)} \\
\text{thus} & ((m, n) \in E(\mathcal{G}(\mathcal{T})) \Rightarrow (\kappa, \sigma) \vDash^{\mathcal{C},[U/x]}_{\mathcal{G}(\mathcal{T}),n} P) \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by } (*)) \\
\text{thus} & (\kappa, \sigma) \vDash^{\mathcal{C},[U/x]}_{\mathcal{G}(\mathcal{T}),n} P \wedge \forall\, U \in S.\ (U, n) \in \sigma & \text{(by } (**)) \\
\text{thus} & (\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} n[P[U/x], S] & \text{(by CFA-Node)}
\end{array}$$

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

Case PPar**.** Then $M = M_1 \parallel M_2$ and $M' = M_1' \parallel M_2'$. We have $(\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} M_1 \parallel M_2$ by assumption, and with CFA-PPar thus $(\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} M_1$ and $(\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} M_2$. 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 $(\kappa, \sigma) \vDash^{\mathcal{C}'_1}_{\mathcal{G}(\mathcal{T})} M'_1$ and $(\kappa, \sigma) \vDash^{\mathcal{C}'_2}_{\mathcal{G}(\mathcal{T})} M'_2$, and the postconditions of the induction hypothesis hold for $\mathcal{C}'_1$ and $\mathcal{C}'_2$. With CFA-PPar we have $(\kappa, \sigma) \vDash^{\mathcal{C}'_1 \cup \mathcal{C}'_2}_{\mathcal{G}(\mathcal{T})} M'_1 \parallel M'_2$, and the postconditions of the theorem hold for $\mathcal{C}'_1 \cup \mathcal{C}'_2$.

Case Struct**.** We have $(\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} M$ 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 $(\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} M$ we have $(\kappa, \sigma) \vDash^{\mathcal{C}''}_{\mathcal{G}(\mathcal{T})} L$, and the postconditions of the induction hypothesis hold for $\mathcal{C}''$. Hence, we can apply the induction hypothesis on $L \xrightarrow{(U,m)\sharp}_{G \in \mathcal{T}} L'$ to have $(\kappa, \sigma) \vDash^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} L'$, and the postconditions of the induction hypothesis hold for $\mathcal{C}'$. By applying Lemma 3.1 (2) again, we have $(\kappa, \sigma) \vDash^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} L'$. □

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 $(\kappa, \sigma) \vDash^{\varnothing}_{\mathcal{G}(\mathcal{T})} N$, then $(T, n) \in \kappa$.*

Proof. We have $(\kappa, \sigma) \vDash^{\varnothing}_{\mathcal{G}(\mathcal{T})} N$ by assumption and can thus apply Theorem 3.3 finitely many times to the derivations $N \longrightarrow^*_{\mathcal{T}} N'$ to yield $(\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N'$. The accumulated postconditions of the applications of Theorem 3.3 give the preconditions for Lemma 3.2. 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 $N \Downarrow^{\mathcal{T}}_n T$ and $(\kappa, \sigma) \vDash^{\varnothing}_{\mathcal{G}(\mathcal{T})} N$, then $(T, n) \in \sigma$.*

Proof. By induction on the shape of the derivation tree for $N \Downarrow^{\mathcal{T}}_n T$, 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:

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

Since we have $(\kappa, \sigma) \vDash^{\varnothing}_{\mathcal{G}(\mathcal{T})} N$ by assumption, we can thus apply Theorem 3.3 finitely many times to the derivations $N \longrightarrow^*_{\mathcal{T}} N'$ to yield $(\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N'$. The accumulated postconditions of the applications of Theorem 3.3 give the preconditions for Lemma 3.1 (2). We can apply Lemma 3.1 (2) to have $(\kappa, \sigma) \vDash^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} N'' \parallel n[\mathsf{store}\ T.P, S]$. Furthermore, by CFA-PPar we have $(\kappa, \sigma) \vDash^{\mathcal{C}'}_{\mathcal{G}(\mathcal{T})} n[\mathsf{store}\ T.P, S]$, 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 [16], 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 [18].

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 to the more "traditional" security protocols for authentication, confidentiality and similar properties. We will then describe SAODV-basic, a simplified version of the combination of the routing protocol AODV [22] and its security extension SAODV [11]. SAODV-basic 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 SAODV-basic 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 the network service can be provided unconditionally, even in an adversarial environment.

Security Modelling Security protocols are usually point-to-point, 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 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 SAODV-basic

The Ad-hoc On-demand Distance Vector (AODV) protocol [22], also standardised by the IETF as RFC 3561 [21], 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 [11], in the process of standardisation by the IETF [10], is a protocol extension of AODV to secure the route discovery mechanism.

Simplifications. While a full description of AODV and its SAODV extension is beyond the scope of this paper, we can illustrate the use of our specification and analysis framework with SAODV-basic, a much simplified version of the combination of the two protocols. A main simplification is obtained by only describing route discovery and no route maintenance; this is straightforward as the two mechanisms are largely independent of each other. More problematic seems our omission of distance information in routing tables and messages. However, one may argue that it should be possible to prove correct the use of direction information independently of distance information. This view is supported by the fact that SAODV uses a distinct second mechanism (hash chains) to deal with distance information. While final conclusions for AODV/SAODV can only be drawn from a full model, we thus have reason to believe that our main results will translate to the full model.

Operation. In Table 8 we show the main subroutines of SAODV-basic, modelled in CBS#. 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:

```
$SendRREQ(dstip, ip) \stackrel{\text{def}}{=}$
  out $\mathit{RREQ}(dstip, ip, ip, \mathit{PubKey}(ip),$
    $\mathit{Sign}(\mathit{PrivKey}(ip), \mathit{Tuple}(RREQ, dstip, ip, \mathit{PubKey}(ip))))$

$ReceiveRREQ(ip) \stackrel{\text{def}}{=}$
  in $\mathit{RREQ}(; dstip, origip, sndip, pubkey, sig).$
    case $pubkey$ of $\mathit{PubKey}(origip;)$
      checksig $sig$ $pubkey$ $\mathit{Tuple}(RREQ, dstip, origip, pubkey)$
        store $\mathit{Route}(origip, ip, sndip).$
          if $dstip = ip$ then
            out $\mathit{RREP}(dstip, origip, sndip, ip, \mathit{PubKey}(ip),$
            $\mathit{Sign}(\mathit{PrivKey}(ip), \mathit{Tuple}(RREP, dstip, origip, \mathit{PubKey}(ip))))$
          else
            out $\mathit{RREQ}(dstip, origip, ip, pubkey, sig)$
      else nil
    else nil
  | $ReceiveRREQ(ip)$

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

Table 8: Subroutines of SAODV-basic specified in CBS#

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 $\mathit{RREQ}(n_d, n_s, n_s, \mathit{PubKey}(n_s), sig)$, where sig is $\mathit{Sign}(\mathit{PrivKey}(n_s), \mathit{Tuple}(RREQ, n_d, n_s, \mathit{PubKey}(n_s)))$.

Receiving RREQs Every node n receiving a route request $\mathit{RREQ}(n_d, n_s, n_i, pubkey, sig)$

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 $\mathsf{Tuple}(RREQ, n_d, n_s, pubkey)$ 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 $\mathsf{Route}(n_s, n, n_i)$ 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. $n_d = n$. 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. $\mathsf{RREP}(n_d, n_s, n_i, n_d, \mathsf{PubKey}(n_d), sig')$, where sig' is $\mathsf{Sign}(\mathsf{PrivKey}(n_d), \mathsf{Tuple}(RREP, n_d, n_s, \mathsf{PubKey}(n_d)))$

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

Receiving RREPs Every node n receiving a route reply message $\mathsf{RREP}(n_d, n_s, n_a, n_i, pubkey', sig')$ checks whether it is the addressee, i.e. $n_a = n$. 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 $\mathsf{Tuple}(RREP, n_d, n_s, pubkey')$. 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, $\mathsf{Route}(n_d, n, n_i)$.

If n was the initiator of the $RREQ$ in the first place, i.e. $n_s = n$, it can now start sending data packets to n_d. Otherwise, n retrieves the route table entry $\mathsf{Route}(n_s, n, n_x)$ 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 $\mathsf{RREP}(n_d, n_s, n_x, n, pubkey', sig')$.

4.3 Security Model for Routing Protocols

From the previous section, we can draw the following conclusions for routing table based protocols such as SAODV-basic: 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 $c_{\mathcal{T}} : \mathsf{T} \to \mathsf{T}$ such that $c_{\mathcal{T}}(U)$ 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 SAODV-basic.

Definition 4.1 (Consistency Mediator for SAODV-basic)

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

Recall that $\mathit{Route}(n_d, n_1, n_2)$ means in SAODV-basic 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 $c_{\mathcal{T}}$ the consistency mediator for N. N is said to be *topology consistent* if the equivalence $N \simeq_{\mathcal{T}}^{c_{\mathcal{T}}} (N \parallel M)$ holds for any network M.

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 mediators c with $c(U) \in \{\varepsilon, U\}$ for all U, the following implication holds: If $(\kappa, \sigma) \vDash^{\mathcal{C}}_{\mathcal{G}(\mathcal{T})} N \parallel M$ and $\forall\ (U, m) \in \sigma.\ c(U) = \varepsilon$, then $N \simeq^{c}_{\mathcal{T}} N \parallel M$.*

Proof. By Definition 2.12, we know $N \simeq^{c}_{\mathcal{T}} N \parallel M$ iff $N \sqsubseteq^{c}_{\mathcal{T}} N \parallel M$ and $N \parallel M \sqsubseteq^{c}_{\mathcal{T}} N$. Since $c(U) \in \{\varepsilon, U\}$ for all U by assumption, the first inclusion can be proved by Theorem 2.14.

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

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

To prove this by contradiction, we assume its negation:

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

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

4.3.3 Attackers

Assume the set of function symbols $\mathcal{F}$ contains the following subsets:

$\mathcal{F}_{Com}$	message constructors
$\mathcal{F}_{2way}$	2-way functions
$\mathcal{F}_{Gen}$	key generation functions

For example, $\mathsf{RREQ}, \mathsf{RREP} \in \mathcal{F}_{Com}$, $\mathsf{Tuple} \in \mathcal{F}_{2way}$ but $\mathsf{Sign} \notin \mathcal{F}_{2way}$. $\mathcal{F}_{Gen}$ contains functions which model key generation function for secret keys, thus $\mathsf{PrivKey} \in \mathcal{F}_{Gen}$. They are treated separately, because they are part of our cryptographic modelling for a public key infrastructure: to a node with IP address ip we are simply associating the public-private key pair $(\mathsf{PubKey}(ip), \mathsf{PrivKey}(ip))$. Obtaining a public key for ip means applying PubKey to ip. At the same time, the attacker should not be allowed to apply the private key function.

We can then define an attacker process P_a as follows:

$$\begin{aligned}
P_1(\mathsf{F}) &\overset{\text{def}}{=} \mathsf{in}\ \mathsf{F}(;x_1...x_{arity(\mathsf{F})}).\mathsf{store}\ x_1....\mathsf{store}\ x_{arity(\mathsf{F})}\\
P_2(\mathsf{F}) &\overset{\text{def}}{=} \mathsf{read}\ x_1....\mathsf{read}\ x_{arity(\mathsf{F})}.\mathsf{out}\ \mathsf{F}(;x_1...x_{arity(\mathsf{F})})\\
P_3(\mathsf{F}) &\overset{\text{def}}{=}\ \ \mathsf{read}\ \mathsf{F}(;x_1...x_{arity(\mathsf{F})})\ \mathsf{store}\ x_1....\mathsf{store}\ x_{arity(\mathsf{F})}\ \mathsf{else\ nil}\\
P_4(\mathsf{F}) &\overset{\text{def}}{=} \mathsf{read}\ x_1....\mathsf{read}\ x_{arity(\mathsf{F})}.\mathsf{store}\ \mathsf{F}(;x_1...x_{arity(\mathsf{F})})\\
P_5 &\overset{\text{def}}{=} \mathsf{read}\ x_1.\mathsf{read}\ x_2.\mathsf{symdec}\ x_1\ x_2\ x\ \mathsf{store}\ x\ \mathsf{else\ nil}\\
P_6 &\overset{\text{def}}{=} \mathsf{read}\ x_1.\mathsf{read}\ x_2.\mathsf{asymdec}\ x_1\ x_2\ x\ \mathsf{store}\ x\ \mathsf{else\ nil}\\
P_a &\overset{\text{def}}{=} (|_{\mathsf{F}\in\mathcal{F}_{Com}}\ P_1(\mathsf{F}) \mid P_2(\mathsf{F})) \mid (|_{\mathsf{F}\in\mathcal{F}_{2way}}\ P_3(\mathsf{F})) \mid\\
&\quad (|_{\mathsf{F}\in\mathcal{F}-\mathcal{F}_{Gen}}\ P_4(\mathsf{F})) \mid P_5 \mid P_6
\end{aligned}$$

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.

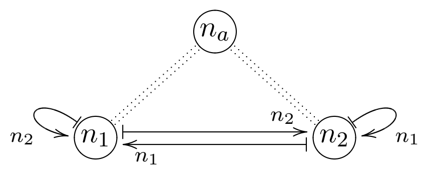
Note that P_1, P_2 range only over functions from $\mathcal{F}_{Com}$, as this is a simple but secure way to limit the number of terms which have to be generated by the analysis.

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 SAODV-basic

The analysis of SAODV-basic 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:

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



$$\begin{array}{llll} \text{At } n_1: & \mathsf{Route}(n_2, n_1, n_2) & \text{At } n_2: & \mathsf{Route}(n_1, n_2, n_2) \\ & \mathsf{Route}(n_2, n_1, n_1) & & \mathsf{Route}(n_1, n_2, n_1) \end{array}$$

Figure 1: Attacker induces topology inconsistency

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

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

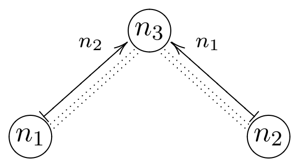
where P_a is defined as in Section 4.3.3 and represents the attack process. Furthermore, a network topology $\mathcal{T}$ is defined such that $E(\mathcal{G}(\mathcal{T})) = \{(n_1, n_a), (n_2, n_a)\}$, thus n_1 and n_2 are never directly connected.

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



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

Figure 2: Topologically consistent network of honest nodes

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] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 104–115. ACM Press, 2001.

[2] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.

[3] Karthikeyan Bhargavan, Davor Obradovic, and Carl A. Gunter. Formal verification of standards for distance vector routing protocols. *J. ACM*, 49(4):538–576, 2002.

[4] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. In *16th IEEE Computer Security Foundations Workshop (CSFW'03)*, pages 126–140, Pacific Grove, California, June 2003.

[5] Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Control flow analysis for the pi-calculus. In *CONCUR '98: Proceedings of the 9th International Conference on Concurrency Theory*, pages 84–98. Springer-Verlag, 1998.

[6] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1), 1990.

[7] Sibusisiwe Chiyangwa and Marta Kwiatkowska. An Analysis of Timed Properties of AODV. In *Proc. 7th IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'05)*, 2005.

[8] Christian Ene and Traian Muntean. A broadcast-based calculus for communicating systems. In *6th International Workshop on Formal Methods for Parallel Programming: Theory and Applications*, San Francisco, 2001.

[9] F. Javier Thayer Fabrega, Jonathan Herzog, and Josua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, pages 191–230, 1999.

[10] Manel Guerrero Zapata. Secure Ad hoc On-Demand Distance Vector (SAODV) Routing. IETF MANET Internet Draft, March 2005.

[11] Manel Guerrero Zapata and N. Asokan. Securing Ad-Hoc Routing Protocols. In *Proceedings of the 2002 ACM Workshop on Wireless Security (WiSe 2002)*, pages 1–10, 2002.

[12] Y. Hu, A. Perrig, and D. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *The 8th ACM International Conference on Mobile Computing and Networking*, 2002.

[13] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[14] Sebastian Nanz and Chris Hankin. Static analysis of routing protocols for ad-hoc networks. In *Proceedings of the 2004 ACM SIGPLAN and IFIP WG 1.7 Workshop on Issues in the Theory of Security (WITS'04)*, pages 141–152, 2004.

[15] Sebastian Nanz and Chris Hankin. Formal security analysis for ad-hoc networks. In *Proceedings of the 2004 Workshop on Views on Designing Complex Architectures (VODCA'04)*, Electronic Notes in Theoretical Computer Science, 2005.

[16] F. Nielson, H. Riis Nielson, H. Sun, M. Buchholtz, R. Rydhof Hansen, H. Pilegaard, and H. Seidl. The Succinct Solver Suite. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *Lecture Notes in Computer Science*, pages 251–265. Springer, 2003.

[17] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis.* Springer, 1999.

[18] H. R. Nielson, F. Nielson, and H. Pilegaard. Spatial analysis of bioambients. In *Static Analysis Symposium (SAS'04)*, volume 3148 of *Lecture Notes in Computer Science*, pages 69–83. Springer Verlag, 2004.

[19] Hanne Riis Nielson and Flemming Nielson. Flow logic: a multi-paradigmatic approach to static analysis. *The essence of computation: complexity, analysis, transformation*, pages 223–244, 2002.

[20] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[21] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. IETF RFC 3561, July 2003.

[22] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc On Demand Distance Vector Routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, 1999.

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

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

[25] Kimaya Sanzgiri, Bridget Dahill, Brian Neil Levine, Clay Shields, and Elizabeth M. Belding-Royer. A secure routing protocol for ad hoc networks. In *10th IEEE International Conference on Network Protocols (ICNP'02)*, Paris, France, 2002.

[26] O. Shivers. Control flow analysis in scheme. In *PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, pages 164–174, New York, NY, USA, 1988. ACM Press.

[27] Oskar Wibling, Joachim Parrow, and Arnold Pears. Automatized verification of ad hoc routing protocols. In *24th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2004)*, LNCS. Springer, 2004.

[28] Irfan Zakiuddin, Michael Goldsmith, Paul Whittaker, and Paul Gardiner. A Methodology for Model-Checking Ad-hoc Networks. In *Model Checking Software: 10th International SPIN Workshop*, volume 2648 of *LNCS*, pages 181–196. Springer-Verlag, 2003.