University of London
Imperial College of Science, Technology and Medicine
Department of Computing

Specification and Security Analysis of Mobile Ad-Hoc Networks

Sebastian Nanz

May 2006

Submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

U.B.L. LONDON UNIV.

I certify that this dissertation, and the research to which it refers, are the product of my own work, and that any ideas or quotations from the work of others are properly acknowledged.

Signed: Sebastian Dans

Date: 16.9.2006

Abstract

Mobile ad-hoc networks consist of mobile wireless devices which autonomously organise their communication infrastructure. Because of the simple network deployment this networking paradigm offers much convenience, but security turns out to be an important concern when considering the threats implied in using the wireless medium. In order to eliminate such concerns, formal specification and analysis techniques have to be used so that the employed communication protocols can be proved secure or their vulnerabilities exposed. While many such frameworks have been proposed for the analysis of classical security protocols, the challenges of the new setting prevent these from being applied directly. 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 this dissertation we present a framework for modelling and reasoning about mobile ad-hoc networks and their security needs. Communication protocols are specified using a broadcast process calculus which makes a clear distinction between the protocol processes and the network's connectivity graph, which may change independently from protocol actions. This modelling formalism is complemented by a control flow analysis which yields an overapproximation of the possible behaviour of a network specified with the calculus. The results obtained from the analysis can be used to establish a behavioural equivalence for networks which in turn is used to specify security requirements. We illustrate the usability of the framework by analysing real-world routing protocols for mobile ad-hoc networks using a prototype implementation of the analysis.

Acknowledgements

First and foremost, I am grateful to Chris Hankin for guiding this work and for having the courage to let me develop my own directions. He has supported me throughout with excellent advice, comments, and suggestions. I would like to thank him for this and our many enjoyable meetings and discussions together.

I also thank the members of the theory group at Imperial for providing me with a friendly and stimulating research environment. Herbert Wiklicky has supported me on many occasions, has given helpful comments, and pointed out related work. Michael Huth and Philippa Gardner have greatly encouraged me with their continuous interest in my work and by discussing aspects of this dissertation with me.

A very special thanks goes to three people who have supported me on the way to my dissertation. Tobias Nipkow has been an invaluable source of support and encouragement during my entire undergraduate studies and has helped to build up my confidence towards starting a PhD. Helmut Schwichtenberg has been a good mentor during my time as a fellow of Studienstiftung des deutschen Volkes, Germany, and I would like to thank him for good advice and interesting discussions during many delightful outings in Munich and the Bavarian alpine uplands. Carsten Schürmann has been a great host and teacher during my stay at Yale, a year which proved to be vital for the development of my research interests.

Furthermore, I had fruitful discussions about the material presented in this dissertation and have received comments and suggestions from: Marta Kwiatkowska and her group at Birmingham, Hanne Riis Nielson and Flemming Nielson and their group at DTU, and K. V. S. Prasad.

I gratefully acknowledge that this work has been supported by a doctoral fellowship of Deutscher Akademischer Austauschdienst (DAAD), Germany.

Finally, I would like to thank my parents, my brother Christian, and Miriam for their love and support.

Contents

1 Introduction **10**

1.1 Background: Mobile Ad-Hoc Networks 11

1.1.1 Characteristics 12

1.1.2 Routing Techniques 14

1.1.3 Example: Ad-Hoc On-Demand Distance Vector Routing 16

1.1.4 Security Issues 18

1.2 Outline of the Dissertation 19

2 Related Work **21**

2.1 Algebraic Process Calculi 21

2.1.1 Background: Foundational Calculi 24

2.1.2 Broadcast Calculi 26

2.1.3 Calculi with Distribution 27

2.1.4 Calculi for the Analysis of Security Protocols 29

2.2 Protocol Analysis Techniques 30

2.2.1 Analysis of Security Protocols 31

2.2.2 Analysis of Communication Protocols 36

2.3 Security of Mobile Ad-Hoc Networks 39

2.3.1 Background: Fundamental Routing Protocols 39

2.3.2 Background: Secure Routing Protocols 40

2.3.3 Security Properties 41

2.3.4 Attacker Models 42

3 The Calculus CBS$^\sharp$ **44**

3.1 Syntax 44

3.2 Operational Semantics 50

3.2.1 Connectivity Graphs 50

3.2.2 Network Topologies 51

3.2.3 Transition Relations 52

3.3 Mediated Equivalence 58

3.4 Notation 62

3.4.1 Syntactic Translations 62
3.4.2 Cryptographic Modelling 63

4 Control Flow Analysis of CBS$^\sharp$ 66
4.1 Background: Control Flow Analysis 66
4.2 Specification 69
4.2.1 Network Topology Abstractions 69
4.2.2 Judgements 70
4.3 Correctness 74
4.4 Properties 76
4.4.1 Characterisation of the Analysis Estimate 76
4.4.2 Relation to the Behavioural Semantics of CBS$^\sharp$ 77
4.5 Implementation 78
4.5.1 Background: The Succinct Solver 79
4.5.2 Finite Analysis of Labelled CBS$^\sharp$ 80
4.5.3 Correctness 84
4.5.4 Constraint Generation Algorithm 85
4.5.5 Precision and Complexity 87

5 Security of Mobile Ad-Hoc Networks 94
5.1 Attackers 95
5.1.1 The Parametric Attacker 96
5.1.2 Attack Scenarios 99
5.1.3 A Type Discipline for Attacker Implementations 100
5.2 A Consistency Condition for Routing Networks 104
5.2.1 Topology Consistency 105
5.2.2 Automated Consistency Checking 107
5.3 Examples: Security Analysis 108
5.3.1 Four Toy Protocols 109
5.3.2 Alternative Attack Scenarios 116

6 Case Studies 119
6.1 SAODV 119
6.1.1 Route Discovery: Digital Signatures 120
6.1.2 Route Discovery: Hash Chains 125
6.1.3 Route Maintenance 129
6.2 ARAN 133
6.2.1 Route Discovery 133
6.2.2 Route Maintenance 137

7 Conclusion **140**
7.1 Review of Research Contributions 140
7.2 Directions for Future Work 143
7.2.1 Quantitative Network Topology Abstractions 143
7.2.2 True Mobility 145
7.2.3 Cut-Offs for Network Topologies 146

A Proofs **147**
A.1 Semantic Correctness of the Control Flow Analysis 147
A.2 Correctness of the Finite Analysis 151

B Protocol Models **155**
B.1 μSAODV 155
B.1.1 μSAODV_RD_DS 155
B.1.2 μSAODV_RD_HC 156
B.1.3 μSAODV_RM 157
B.2 μARAN 159
B.2.1 μARAN_RD 159
B.2.2 μARAN_RM 160

Bibliography **163**

List of Tables

1.1 Mobile ad-hoc networks: protocol architecture 14

3.1 Syntax of CBS$^\sharp$ 45
3.2 Notation for meta-variables 46
3.3 Free variables 49
3.4 Labelled transition relation 54
3.5 Reduction relation 56
3.6 Structural equivalence 57
3.7 Barb predicate 59
3.8 Convergence predicate 60

4.1 Control Flow Analysis for CBS$^\sharp$ 72
4.4 Finite Analysis for Labelled CBS$^\sharp$ 83
4.5 Constraint Generation Algorithm 86
4.6 Variant Analysis for CBS$^\sharp$ 89
4.7 Running time measurements 93

5.1 The parametric attacker 98
5.2 Typed Finite Analysis for Labelled CBS$^\sharp$ 101
5.3 Typed Constraint Generation Algorithm 102
5.4 Analysis estimate: Protocol 1 111
5.5 Analysis estimate: Protocol 2 113
5.6 Analysis estimate: Protocol 3 115
5.7 Analysis estimate: Protocol 4 116
5.8 Analysis estimate: variant scenario for Protocol 4 117

List of Figures

1.1 Mobile ad-hoc networks: connectivity graphs 12
1.2 Mobile ad-hoc networks: protocol reference model 13
1.3 AODV route discovery: initialisation and reverse route setup 17
1.4 AODV route discovery: forward route setup 18

3.1 Example 3.5: connectivity graph 51
3.2 Example 3.8: network topology 52

5.1 Example 5.1: parametric attacker 97
5.2 The Dolev-Yao attacker as a boundary case of the parametric attacker . 99
5.3 Protocols 1–4: network topology 109
5.4 Protocol 1: routing loop attack 112
5.5 Protocol 2: grey hole attack 114

6.1 μSAODV_RD_DS: abstract connectivity graph 123
6.2 μSAODV_RD_DS: topology inconsistency 124
6.3 μSAODV_RD_HC: abstract connectivity graph 128
6.4 μSAODV_RD_HC: wormhole attack 128
6.5 μSAODV_RM: network connectivity graph and routing table entries . . 132
6.6 μSAODV_RM: route invalidations 133
6.7 μARAN_RM: network connectivity graph and routing table entries . . . 139

7.1 Example: network topology 144
7.2 Two-valued and many-valued network topology abstractions 144

Chapter 1

Introduction

Mobile computing, i.e. the use of portable computers with networking capabilities, as well as the advances in wireless technology have revolutionised the way computers are used. Increasingly, users take it for granted that access to networking services is provided anytime and anywhere. However, today's networks rely to a large degree on some kind of fixed infrastructure, such as wireless access points, which cannot be available in all situations. Mobile ad-hoc networks address this problem by allowing mobile wireless devices to form temporary multi-hop networks which might either operate in a stand-alone fashion or be used to extend the range of an access point.

This networking approach relies on the cooperation of nodes in the ad-hoc network: each node assumes the functions of a router and relays packets on paths to other nodes. Finding these paths in an a priori unknown and constantly changing network topology is the crucial functionality of any ad-hoc network and is solved by specifically designed routing protocols operating at the network layer. Other protocol layers and, eventually, network programs are thought to work with supposedly minor modifications once the routing protocols are in place. Thus, mobile ad-hoc networks promise the convenience of infrastructureless networking and a rapid network deployment at the cost of sharing resources by participating nodes.

However, the increased convenience of mobile ad-hoc networks introduces additional security hazards as well. Using the wireless medium itself is a cause of natural concern since this medium is far easier to access by intruders than wired connections. Some of the implied threats can be addressed by security protocols which, for instance, might ensure end-to-end confidentiality and authentication when employed at one of the upper layers of the protocol stack, e.g. the transport layer. Nonetheless, at the network layer, where mobile ad-hoc networks are most vulnerable, an intruder is then still able to interfere with the routing service – without compromising end-to-end confidentiality and authentication, but rendering the network utterly unusable. Secure versions of routing protocols are developed to counter such attacks and thus to ensure that mobile ad-hoc networks can work even in an adversary environment.

While the dynamics and tight resource constraints of mobile ad-hoc networks make it difficult to develop efficient routing protocols even under the assumption of a friendly environment, the security aspect makes it all the more challenging. It might be seen as a strong confirmation of this claim that the related area of security protocols has a long history of discovery of flaws in fallaciously simple looking protocols. Formal specification, analysis, and verification have thus been recognised as a necessity in this area, and many effective frameworks have been proposed to achieve this task. Of course, there are additional reasons why security protocols have become a favourite target of the formal-methods community: they are typically concise and easily expressible, tied by definition to some desired security property, and pose many distinctly diverse challenges which make them ideal objects for studying security problems and the development of versatile tools. Secure communication protocols on the other hand are large, and often developed without a clear notion of the security requirements. The challenge for formal methods is to prove whether they can scale up to this protocol class and help clarify some aspects of its security.

Thesis Statement. The thesis proposed by this dissertation is that formal specification and security analysis techniques can and should be used to address the challenges posed by the analysis of secure communication protocols and to produce automated analysis tools which are valuable and intelligible even for protocol developers normally pursuing a non-formal approach. To justify this thesis, we focus on secure routing protocols for mobile ad-hoc networks as an interesting subclass of this family of network protocols. The main challenges are: finding a suitable representation of the broadcast and multi-hop behaviour of mobile ad-hoc networks; finding a suitable abstraction of this behaviour in order to automate their analysis; and, formalising and expressing a suitable threat model and security property in this setting. We address these challenges and propose a specification and analysis framework, the usability of which is demonstrated by analysing real-world secure routing protocols for mobile ad-hoc networks using a proof-of-concept implementation.

In the remainder of this chapter, we first present the fundamental concepts and characteristics of mobile ad-hoc networks in §1.1, to serve as background information for all of the following chapters. We then give an outline of the dissertation in §1.2.

1.1 Background: Mobile Ad-Hoc Networks

The idea of mobile, wireless, multi-hop networks operating without pre-existing infrastructure dates back to the concept of multi-hop *Packet Radio NETworks (PRNETs)* in the early 1970s and has almost continuously been researched up until today [132, 82]. Especially during the late 1990s, in the wake of the technological advancements which

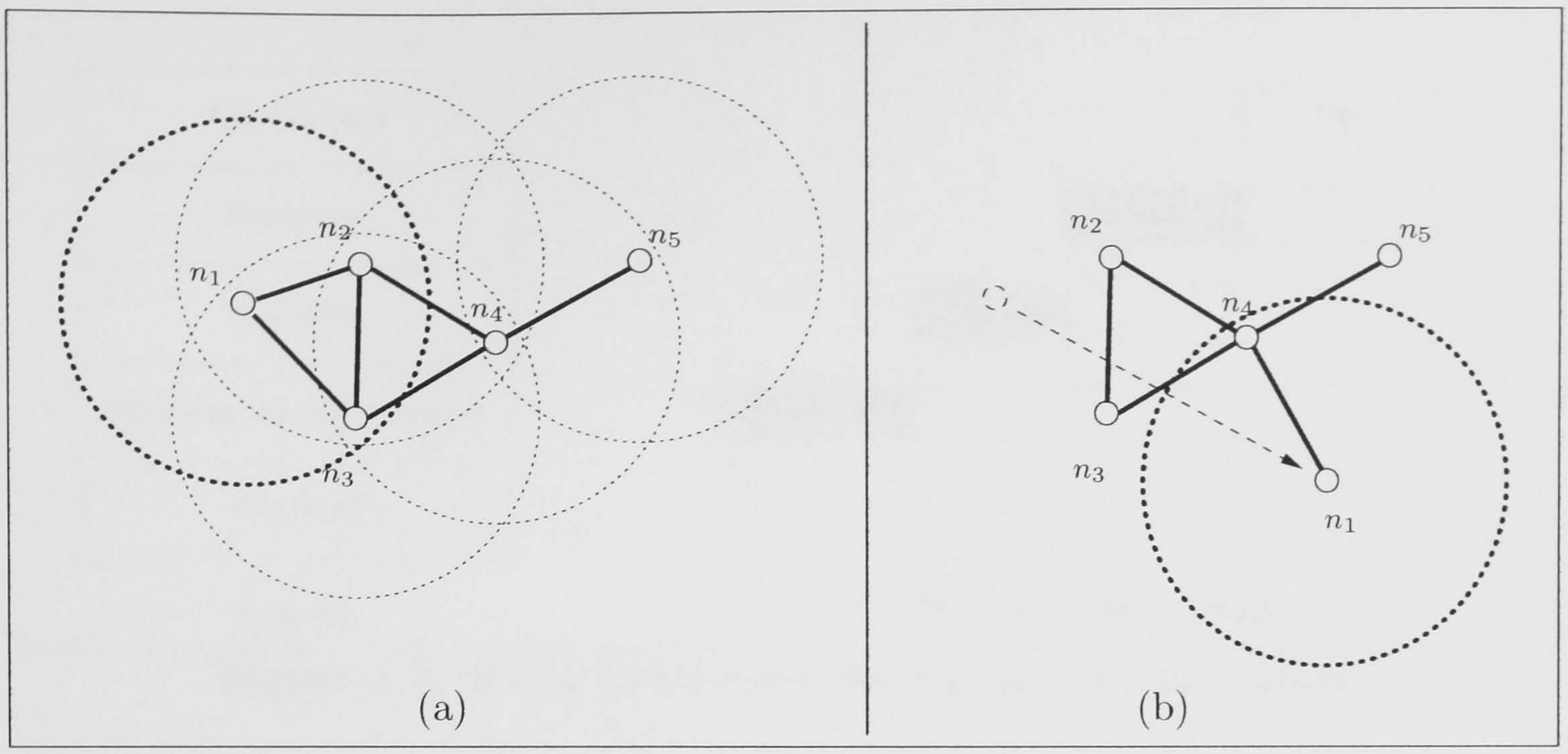

Figure 1.1: Mobile ad-hoc networks: connectivity graphs

made portable computers and wireless hardware available at low cost, research activity on this topic has increased dramatically. This rise in interest is also documented by the fact that the *Internet Engineering Task Force (IETF)* has established a working group (IETF MANET WG) in order to standardise IP routing protocol functionality for this networking approach.

This class of networks is now most commonly known as *Mobile Ad-hoc NETworks (MANETs)*. Two other terms are used to denote network classes that bear strong similarities with this concept: *Wireless Mesh Networks* work in a similar peer-to-peer fashion, but need not be mobile and put an emphasis on redundant node connectivity (the *mesh topology*), to achieve particular robustness; *Sensor Networks* consist of a (potentially large) number of distributed devices which use sensors to monitor environment conditions. A wealth of commercially interesting applications has been identified as well, some of which can be found in [114]. However, up until now, none of the competing protocol proposals for the realisation of mobile ad-hoc networks has yet left the stage of simulation and small-scale real-world experiment [82].

1.1.1 Characteristics

Mobile ad-hoc networks consist of mobile computing devices communicating via wireless multi-hop transmissions. These devices are frequently denoted by the more abstract term *nodes* since they are the nodes (vertices) in the connectivity graph determined by their transmission ranges and physical location. Informally, this graph is called the *network topology*. Figure 1.1 (a) shows an example: there, an edge is drawn between two nodes whenever they are able to communicate (we assume bidirectional links for the moment). Movements of nodes dynamically change the appearance of this graph, as depicted in Figure 1.1 (b) for the node n_1.

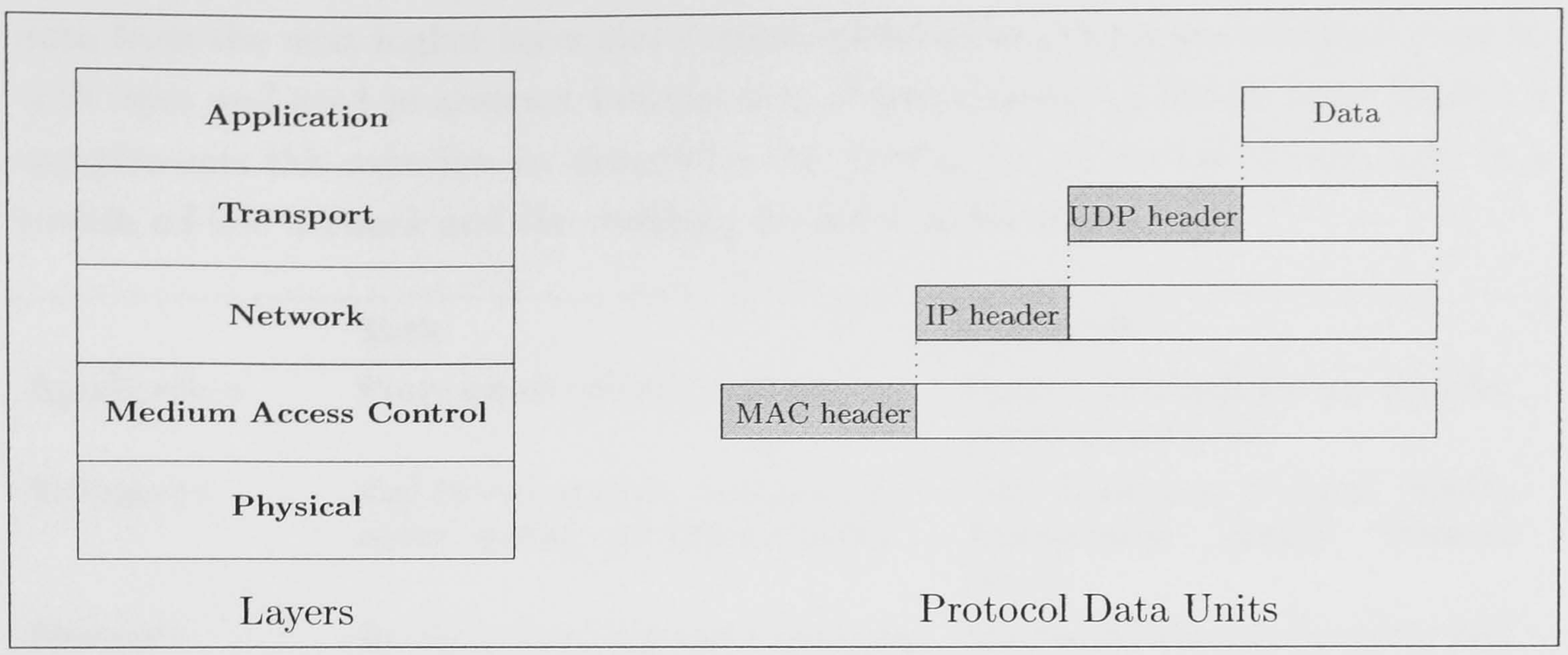

Figure 1.2: Mobile ad-hoc networks: protocol reference model

The most prominent characteristics enjoyed by all mobile ad-hoc networks can thus be succinctly summarised as follows:

Mobile Nodes are free to move about arbitrarily. Therefore network connectivity may change over time as it depends on the nodes' physical location.

Wireless Nodes are equipped with wireless transmitters and receivers. Network connectivity is therefore determined by the relative positions of the nodes and their transmission ranges, which may vary from node to node.

Multi-hop If two nodes are not within direct transmission range of each other but connected by a path in the connectivity graph, they can still communicate by using nodes on the path as intermediate hops. Thus, unlike other networking paradigms which have dedicated routers, every node may act as both host and router.

The following further characteristics [38] concerning the technical realisation of ad-hoc networks are a prerequisite for understanding the difficulty of designing protocols in this setting:

Bandwidth constraints Wireless links have significantly lower capacity than their hardwired counterparts and their performance can be highly variable.

Energy constraints Some or all nodes rely on exhaustible means for their energy.

As in all communication networks, the desired behaviour of mobile ad-hoc networks is realised by a number of protocols building on each other. The relevant design issues can thus best be explained by looking at a protocol reference model such as the one in Figure 1.2. It shows the layers of a general 5-tier architecture as it may be found in any networking textbook, e.g. [131], and the *Protocol Data Units (PDUs)* (combination of

data from the next higher layer and control information) which are communicated by each layer and used to abstract functionality of lower layers for higher ones. Table 1.1 complements this overview by describing the tasks to be performed at each layer in a mobile ad-hoc network and the resulting protocol architecture.

	Task	Protocols
Application	Provision of end-user services	Various protocols for file transfer, electronic mail, etc.
Transport	End-to-end transfer between processes running on different nodes	User Datagram Protocol (UDP), Transmission Control Protocol (TCP)
Network	Routing of packets under changing topology, including route discovery and route maintenance	Specifically designed routing protocols
Medium Access Control	Coordination of the access of multiple nodes to the shared wireless medium	IEEE 802.11
Physical	Transmission of bits over the wireless medium	Wireless modem hardware

Table 1.1: Mobile ad-hoc networks: protocol architecture

The overview in Table 1.1 shows that the network layer, namely the realisation of route discovery and route maintenance, poses the main challenge for the design of mobile ad-hoc networks. This is not surprising as node mobility implies that the usual routing assumptions of fixed and known neighbours are no longer applicable. Protocols on higher and lower layers of the reference model are expected to work with only minor modifications once the routing problem is solved. In the next section we give an overview of currently proposed routing techniques.

1.1.2 Routing Techniques

In order to understand the main functionality of the network layer, it is useful to make a distinction between two concepts [131]: forwarding and routing. *Forwarding* is the process activated when a packet arrives at a node, in the event of which the node looks up the appropriate routing information, usually stored in a so-called *routing table*, and relays the packet towards its destination. *Routing* is the process responsible for the computation of the routing information, and is defined by what is called the *routing protocol* (or *routing algorithm*). A routing protocol prescribes the exchange of information about the network topology with other nodes and the decision process about the routing information a node finally commits to. Note that a routing protocol, although formally located at the network layer, can make use of protocols at higher layers for the sake of communicating with neighbouring nodes about the network topology. One can thus distinguish between *routing packets*, used by the routing processes, and *data*

packets, which denote all other packets sent on the network.

Three fundamental approaches to routing in mobile ad-hoc networks are taken [114]:

Proactive Nodes continuously exchange information about the network topology and update their routing tables, so that they are constantly aware of available routes.

On-demand (or Reactive) Routes are only looked for and established when they are needed.

Hybrid A combination of the proactive and on-demand techniques is provided.

In general, proactive approaches are thought to have shorter latency, as routes are instantly available, whereas on-demand approaches will have lower routing control overhead and therefore achieve higher data throughput.

Protocols can further be classified according to general routing techniques [131], which are applicable to other networking paradigms besides ad-hoc networks:

Hop-by-hop routing This scheme relies on the fact that every node knows the next hop on the path to a certain destination and the distance to the destination according to some metric (e.g. number of hops or delay). This information is stored in routing tables, and the header of a data packet has to contain information about its destination only. Hop-by-hop routing methods can be categorised into two primary classes:

(1) Link state

(2) Distance vector

In *link state* algorithms, a node periodically sends information about its neighbours to all known nodes. As a node receives this information, it applies a shortest-path algorithm to recompute its routing tables for every destination. In *distance vector* algorithms, e.g. the *Distributed Bellman-Ford (DBF)* algorithm, a node periodically sends the current shortest path estimates for all known nodes to its neighbours. The neighbours use an algorithm to update their own shortest path estimates in their routing tables.

Source routing In this routing paradigm, the sender prescribes the entire path to the destination. Routes are held in so-called *route caches* for future use, appended to the header of sent packets whenever needed, and intermediate nodes determine the next hop solely by examining this information.

These general routing approaches have to be modified in order to apply to mobile ad-hoc networks. In order to illustrate this with an example, we describe an on-demand distance vector routing protocol for mobile ad-hoc networks in the following section.

1.1.3 Example: Ad-Hoc On-Demand Distance Vector Routing

Based on the mentioned fundamental routing approaches, a multitude of protocols for ad-hoc networks have been proposed. In this section we describe one of them as an archetypal example in order to provide a broad idea of their common functionality and to help motivate some of the design decisions of the framework developed in this dissertation.

The Ad-hoc On-demand Distance Vector (AODV) protocol [117] is one of the more prominent routing protocols for mobile ad-hoc networks and has also been standardised by the IETF as RFC 3561 [115]. It consists of two main parts, *route discovery* and *route maintenance*. For the sake of this introductory example, we describe only the former and omit some details. We also strive for making the reader familiar with some of the notation used in this dissertation: for instance, functions IP(.), RREQ(.), etc., applied to a number of components, are used to represent the Protocol Data Units and their fields.

Route Request Initialisation. If a node *src*, called the *source*, needs to communicate with another node *dst*, called the *destination*, *src* initiates a route discovery process by broadcasting a *Route Request (RREQ)* message as follows:

$$\mathsf{IP}(src, bcst, \mathsf{RREQ}(req_id, dst, dst_seq, src, src_seq, 0))$$

This represents an IP packet which carries in its header the IP sender address *src* (node identities and IP addresses are identified) and the IP limited broadcast address 255.255.255.255 (abbreviated by *bcst*) as IP destination address. The body of the IP packet is a RREQ packet consisting of a *Request ID* (*req_id*), the IP addresses of *dst* and *src*, *sequence numbers* (*dst_seq* and *src_seq*), and a *hop count* (i.e. the number of hops between *src* and the current node), initially taken to be 0. The counters *req_id* and *src_seq* are maintained locally by every node and incremented whenever a RREQ is sent, with the purpose of identifying requests and determining the freshness of routes, respectively. The component *dst_seq* is set to the last sequence number *src* has seen for *dst*, or 0.

Reverse Route Setup. Every node *n* receiving a RREQ packet from a node *snd* aborts if it has seen the pair (*req_id*, *src*) already, to prevent processing of duplicates. Otherwise, *n* makes an entry into its routing table in order to provide information about the next hop *snd* on a *reverse route* to *src*:

$$\mathsf{Rte}(src, snd, src_seq, hop_cnt)$$

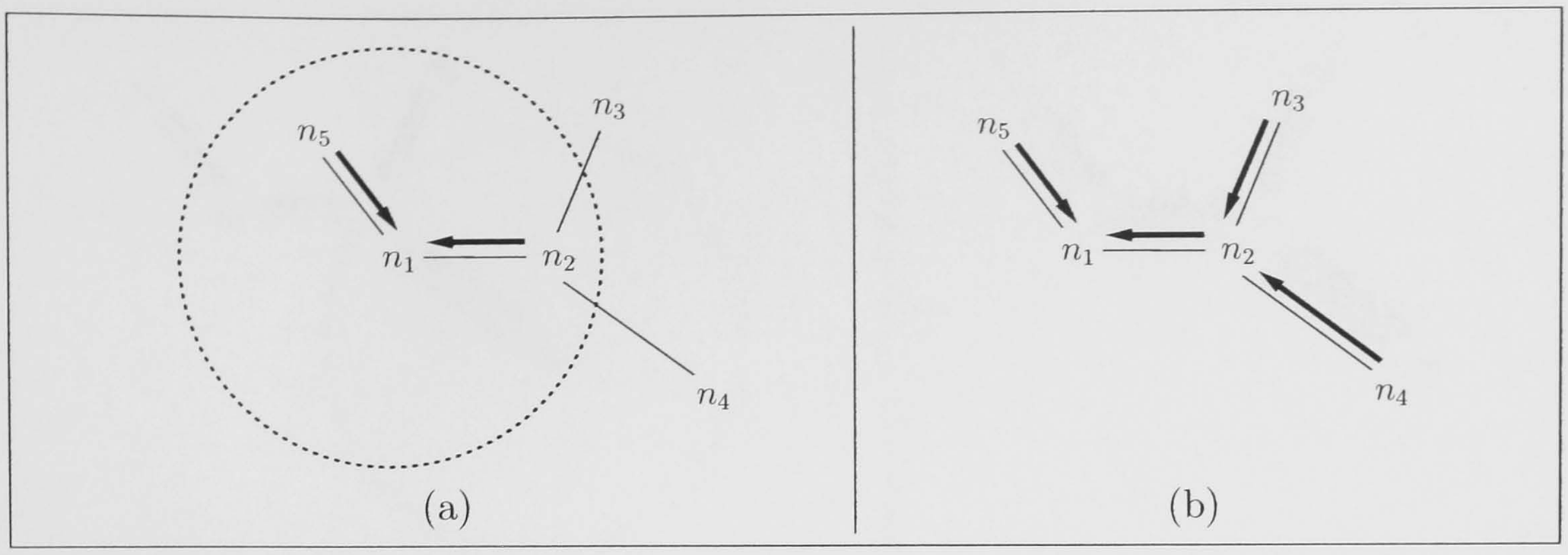

Figure 1.3: AODV route discovery: initialisation and reverse route setup

Node n then checks whether it knows a route to the destination with a destination sequence number at least as great as indicated in the RREQ (n could also be the destination itself). If no, n rebroadcasts the RREQ, incrementing the hop count *hop_cnt* and modifying the IP sender address:

$$\mathsf{IP}(n, bcst, \mathsf{RREQ}(req_id, dst, dst_seq, src, src_seq, hop_cnt + 1))$$

If yes, n sends instead a *Route Reply (RREP)* to the next hop *next* on the reverse route to *src*, using the most recent sequence number dst_seq' associated with *dst* and hop count set to 0:

$$\mathsf{IP}(n, next, \mathsf{RREP}(dst, dst_seq', src, 0))$$

This behaviour is illustrated in Figure 1.3, where node n_1 represents the source and node n_4 the destination: part (a) shows the initial broadcast of the RREQ by n_1 and the setup of the reverse route (arrows) by the adjacent neighbours; part (b) shows all route table entries after complete propagation of the RREQ through the network, in particular the full reverse route between n_4 and n_1.

Forward Route Setup. Every node n receiving a RREP packet from a node *snd* tries to retrieve a routing table entry $\mathsf{Rte}(src, next, src_seq, hop_cnt)$ indicating a next hop *next* on route to *src*. (If unsuccessful, route maintenance has to take place.) Node n makes an entry into its routing table in order to provide a *forward route* to *dst*:

$$\mathsf{Rte}(dst, snd, dst_seq, hop_cnt)$$

Node n then resends the RREP to *next*, except if n was the source of the RREQ in the first place, in which case the route is established.

Figure 1.4 shows an illustration of this forward route setup: part (a) shows the response of the destination n_4 to the RREQ by transmitting a RREP, and the forward route setup by n_2; part (b) shows the full forward route between n_1 and n_4.

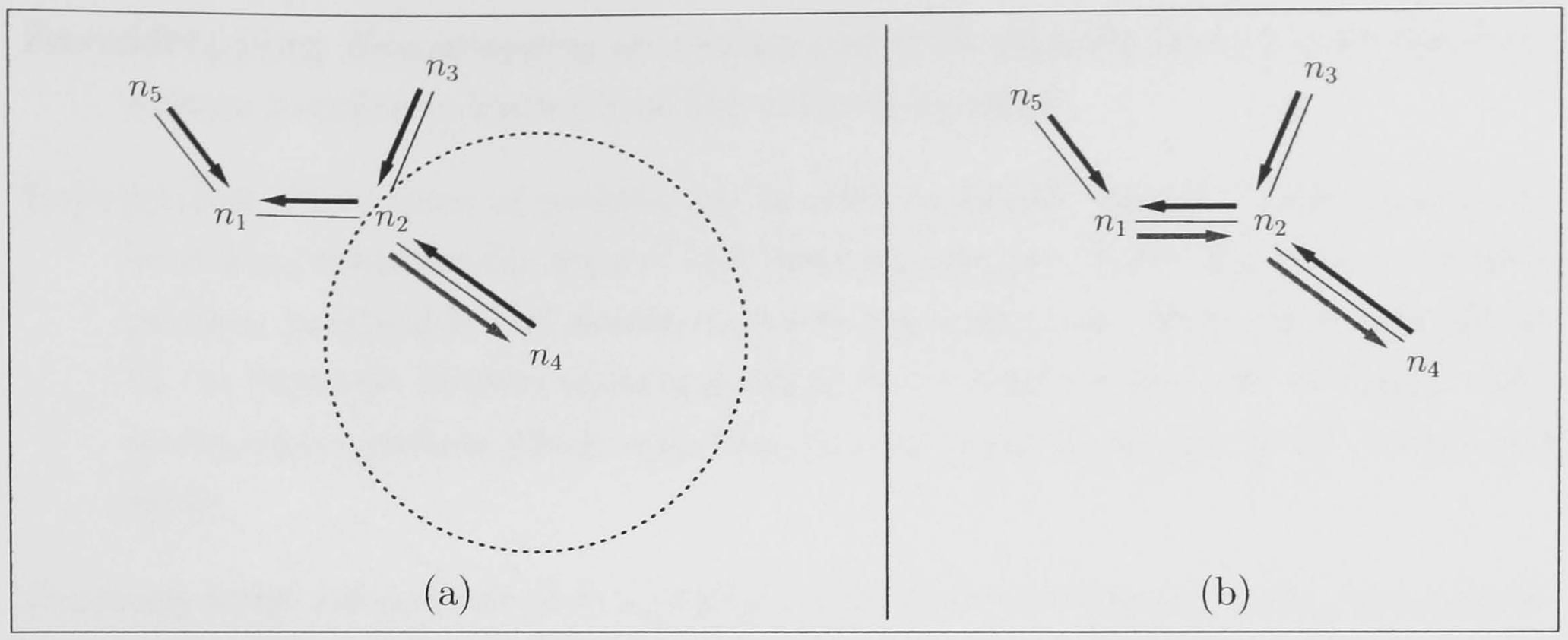

Figure 1.4: AODV route discovery: forward route setup

1.1.4 Security Issues

The protocol AODV, like many of the currently proposed routing protocols for mobile ad-hoc networks, is mainly optimised for performance and is assumed to be employed in a friendly, cooperative environment. More recent research has recognised that this assumption is unrealistic in most situations and that mobile ad-hoc networks pose a number of security issues which have not been dealt with in traditional security design [138]. Ignoring these issues means that fairly powerful attacks can often be launched just by not behaving according to the protocol specifications.

Known attacks have been classified in various ways. In [74], the authors distinguish *routing disruption* (causing data packets to be routed in dysfunctional ways) from *resource consumption* attacks (causing resource drain in the network). In [126], attacks are distinguished by the activities of the attacker, namely modification of message contents, impersonation, and fabrication of routing messages. We find it however most important to stress that *all* these are instances of *denial-of-service (DoS)* attacks, if DoS is understood in its most general meaning as an attack on networks that causes a loss of service to users.

Attack classifications are common in the area of mobile ad-hoc networks. This is not surprising since the existing proposals for secure routing protocols are typically attack-oriented in the sense that they enumerate security threats and design specific countermeasures to thwart them [138]. Such an approach is problematic because one can never be sure to have found all possible attacks. In this dissertation, we pursue a different approach in order to solve this problem: we aim for stating a formal security property and proving that it is fulfilled by a model of the protocol. Nevertheless, in order to find such a property, it is certainly helpful to know about the range of possible attacks, and we describe some of the important ones to conclude our background discussion of mobile ad-hoc networks:

Eavesdropping *Eavesdropping* on routing packets is typically part of more elaborate attacks in order to learn about the network topology.

Fabrication *Fabrication* of packets can be used to launch various attacks, which can be distinguished by the type of the fabricated packet. When injecting extra data packets, bandwidth and power resources are consumed. More dangerous might be the injection of false routing packets, for example in the case of forged route maintenance packets which cause routes to be invalidated, just to be constructed again.

Routing loop Loop-freedom is a crucial property for routing protocols because otherwise data packets might be forwarded in a cycle (the *routing loop*) without reaching the destination, at the cost of bandwidth and power resources. By using *impersonation*, i.e. *spoofing* of IP addresses, attackers can artificially create such loops in insecure networks. A simpler variant of this attack is to cause *detours*, which are suboptimal routes.

Black hole A *black hole* is created by sending forged routing information which causes all packets for a particular destination to be routed to one particular node (e.g. the attacker), which just drops them without forwarding. The attack has been described in particular for AODV, where choosing a sequence number higher than the authentic value will cause nodes to update their routing information to arbitrary values.

Grey hole An attacker can act as a *grey hole* if forwarding packets only selectively. Some authors dismiss such and similar attacks as too difficult to defend against since they are hard to distinguish from Byzantine, i.e. random, failures.

Wormhole A *wormhole* requires a pair of attackers which have established a private connection, either by encapsulating messages and using the functionality of the wireless network, or by using an altogether private network. Using the wormhole, the attackers may, for example, try to falsely represent network distances by short-circuiting the normal packet flow.

1.2 Outline of the Dissertation

As our thesis statement indicates, this dissertation proposes to apply classical formal specification formalisms and analysis techniques to the setting of secure mobile ad-hoc networks. In particular, we base our framework on a process calculus and static analysis techniques which are tailored and further developed for this purpose. Some of this material has been presented in our journal publication [100]. This dissertation is organised as follows:

In Chapter 2 we review related work falling into three areas: algebraic process calculi, protocol analysis techniques, and security of mobile ad-hoc networks. The chapter contains background information as well in order to provide readers unfamiliar with either formal approaches or mobile ad-hoc networks with general ideas and pointers to important works in this area.

In Chapter 3 we define a process calculus CBS$^\sharp$ as a specification formalism for mobile ad-hoc networks. We describe the syntax and operational semantics of this calculus, develop behavioural equivalences, and conclude with techniques for security modelling.

Chapter 4 describes the development of a behavioural analysis for networks specified in CBS$^\sharp$. We first give an overview of control flow analysis, a program analysis technique which we adapt for our needs. We specify an analysis on the terms of CBS$^\sharp$ by way of a so-called flow logic and prove its correctness with respect to the operational semantics and related properties. We transform the analysis to work over a finite set of values and describe an implementation using an off-the-shelf constraint solving tool.

In Chapter 5 we apply the specification and analysis framework to security issues in mobile ad-hoc networks. We define a parametric attacker model suitable in the setting of mobile ad-hoc networks. We then use a behavioural equivalence of CBS$^\sharp$ to express a security condition for routing protocols and show that the framework can be used to automatically derive this condition. We conclude with an instructive series of toy protocols and their analyses to prepare the reader for the case studies in the subsequent chapter.

In Chapter 6 we analyse and compare two approaches to secure hop-by-hop routing in mobile ad-hoc networks. We include the full program text of the protocols as used by our prototype implementation and present the attacks we discovered.

We conclude the dissertation in Chapter 7 with a discussion of research contributions made and directions for future work.

The appendix provides supportive material. In Appendix A two of the longer correctness proofs for Chapter 4 are carried out. Appendix B supplements the full text of the protocols discussed in Chapter 6.

Chapter 2

Related Work

Formal specification and security analysis have a long research history in computer science, and the same is true for the development of network protocols. However, only few works to date have attempted to bring these two strands of research together, let alone in the small realm of mobile ad-hoc networks. A discussion of related work for this dissertation thus consists mainly of a review of works in the separate areas. Whenever felt necessary, we enrich the discussion with general background information which might be beneficial for readers whose main expertise does not cover all presented areas.

This chapter is in three sections. In §2.1 we present work in the area of process algebra, in particular in the area of broadcast calculi, distributed calculi, and calculi for security analysis. Various approaches to protocol analysis for both security protocols and network protocols are discussed in §2.2. In §2.3 we present mostly non-formal work on the security of mobile ad-hoc networks.

2.1 Algebraic Process Calculi

Process algebra is a formal approach to the study of concurrent processes which uses algebraic structures, so-called *process calculi*, to model processes and reason about them. Its algebraic, and thus, more generally speaking, linguistic character distinguishes it most significantly from other concurrency models, such as Petri Nets [118], the Actor Model [68], or Input/Output Automata [88]. In order to understand process algebra, it is helpful to consider the elements which typically constitute a process calculus:

Syntax The shape of the terms of a process calculus is defined inductively.

Formal semantics A process calculus is given a formal semantics to describe the execution capabilities of a system represented as a term. For programming languages, three main styles of formal semantics are usually distinguished: operational, denotational, and axiomatic semantics. These are mirrored in the area of process

calculi. Operational semantics is usually expressed in two styles: *Structural Operational Semantics* [119] generates labelled transition systems so that states are closed terms and state transitions are obtained from a collection of inference rules; and the *Chemical Abstract Machine (CHAM)* [13] style gives the semantics in terms of a set of reduction rules for term rewriting and a structural congruence to rearrange terms. *Denotational semantics* has originally been used to model the meaning of a program as a function that maps input to output, an approach which has been generalised in domain theory. The general idea (mapping programs into domains representing the effect of its execution) has been applied to process calculi as well, for example to CSP [69], where processes are mapped into their sets of traces, failures, and divergences. *Axiomatic Semantics* has been used to define ACP [12], which is described by a set of axioms with respect to a simple process algebra syntax. In yet another approach, *Hennessy-Milner Logic* [65], processes are characterised by formulae of modal logic, which express properties fulfilled by the processes.

Behavioural relations Relations defined on terms allow to compare different systems, usually evidently correct specifications with models of a possible implementation. These relations can be equivalences, to state that processes share the same behaviour, or preorders, to express refinement.

While one of the initial incentives for the development of process calculi was to find a fundamental model of concurrent computation, a "λ-calculus of concurrency", it seems now widely accepted that the nature of processes might be too diverse to be described in a single algebraic approach. In any case, a plethora of different calculi has been defined, rivalling in number the programming languages proposed to date: few of them can be considered fundamental. Although this original goal has not been attained, the success of process algebra can be seen in its capability to match, nonetheless, each of the various aspects of the nature of processes that have been discovered: by many different calculi, each with their own specific applications and reasoning techniques which are interesting for their own sake.

It is thus helpful to try to characterise the nature of processes as this might reveal a possible classification of process calculi. We consider three orthogonal concepts:

Mobility Milner [95] suggests to answer two questions in order to clarify the use of this term: "What kind of entities move?", and "In which space do they move?". In our case, note that the term mobility is to refer to the abstract process model, not to possible applications of this model. It is thus only sensible to talk about an equally abstract space of processes and communication links, in which the mobile entities clearly are either links (1) or processes (2). Two aspects of mobility can thus be considered:

(1) Link mobility

(2) Process mobility

To clarify this point further, in a particular application these aspects can then be used to model mobility in more concrete spaces: in a cellular network, the roaming of mobile phones around base stations can be modelled by link mobility; mobile code sent over the Internet and executed at a remote host can be modelled by process mobility.

While Milner's distinction helps to describe the term mobility as such, note that the particular kind of link mobility developed by Milner in the π-calculus is rather different from the notion we want to establish: π-calculus processes can use new links once they have learnt about them by communication with other processes; we emphasise that the availability of a link in wireless networks is the result of environment conditions which cannot be influenced by process actions.

Distribution *Locations* (or *localities*) can be used to explicitly describe the distribution of processes. They represent units at which processes are grouped according to some criterion, for example describing physical sites at which processes share computation resources.

Communication This distinguishes the way in which processes can interact. The fundamental distinction is between one-to-one (1) and one-to-many communication (2):

(1) Point-to-point

(2) Broadcast

Orthogonally, we can differentiate whether process interaction requires joint participation of all communication partners (a), or can be delayed (b), as in buffered communication:

(a) Synchronous

(b) Asynchronous

More recently, process algebra has branched into areas which no longer fit in such an abstract classification, but have to be described in terms of their application. Examples are calculi for systems biology, probabilistic and stochastic calculi for quantitative evaluation of systems, or calculi for security analysis. Our approach can be seen in this line of research, as it is strongly motivated by the application domain of wireless systems. However, the calculus presented in Chapter 3 can also be viewed in terms of the above classification, namely as a distributed calculus with link mobility and synchronous broadcast communication.

2.1.1 Background: Foundational Calculi

Before describing the process calculi closely related to our approach, we first give a background on what one might call "pure" or "foundational" calculi, which have been developed with the goal of providing very general process models. Most specialised calculi are derived from their seminal ideas.

CCS. The *Calculus of Communicating Systems (CCS)* by Milner [93, 94] provides a static, non-distributed model of processes with synchronous point-to-point communication. Actions $\alpha ::= a \mid \bar{a} \mid \tau$ represent atomic execution steps, internal (silent) in the case of τ, otherwise externally observable. They give rise to labelled transition relation $\xrightarrow{\alpha}$ on processes, constituting the operational semantics of the calculus. Sequential composition (prefix) $\alpha.P$, nondeterministic choice (summation) $P_1 + P_2$, and the terminated process $\mathbf{0}$ have straightforward inference rules. In the case of parallel composition $P_1 \mid P_2$, both branches can execute on their own, or interact internally via handshake:

$$\frac{P_1 \xrightarrow{a} P_1' \qquad P_2 \xrightarrow{\bar{a}} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$$

Restriction $P \backslash K$ expresses that actions contained in K can only be used for internal communications within P, and renaming yields a process $P[f]$ in which all actions α of P are relabelled $f(\alpha)$. Recursion is achieved by defining and invoking constants A via equations $A \stackrel{\text{def}}{=} P$, where A can occur recursively in P.

A value-passing variant of CCS introduces expressions e, values v, and variables x, and understands actions as channels through which values can pass. Prefixes are redefined to $a(x) \mid \bar{a}(e) \mid \tau$, and evaluated using the following new rules:

$$\bar{a}(e).P \xrightarrow{\bar{a}(v)} P, \text{ if } e \text{ evaluates to } v \qquad\qquad a(x).P \xrightarrow{a(v)} P[v/x]$$

It turns out that the value-passing variant can be encoded in the basic calculus.

Many behavioural relations have been used for CCS, including trace- and testing-based notions. We want to highlight two of the equivalences employed originally in [94]: strong equivalence and observation equivalence. Strong equivalence is based upon the notion of *strong bisimulation*, i.e. every action of one agent must be matched by the other, and turns out to be a *congruence*, thus preserved under all calculus operations. Observation equivalence relies on *weak bisimulation*, where τ-actions are ignored.

CSP. Hoare [69, 70] and Brookes, Hoare, and Roscoe [25] have proposed *Communicating Sequential Processes (CSP)*, providing a process model similar to CCS. Differences are CSP's two concepts of choice (external choice of the environment and nondeterministic choice) and the fact that the parallel composition operator does not hide communications and thus allows any number of processes to participate in the same event.

The operational semantics of CSP is usually expressed in denotational style, such that a process is defined by the pair of its failures (traces after which a process is stuck) and divergences (traces after which a process behaves chaotically). Behavioural reasoning is another important distinction from CCS: while CCS relies on equivalences, meaning that an implementation and its specification must be essentially indistinguishable, CSP uses refinement, where the observations of a correct implementation only have to be included in those of its specification, and not vice versa. The asymmetry of the refinement relation allows a specification to be more abstract, which can be advantageous for incremental development of systems.

ACP. Bergstra and Klop [12] (see also [9]) have presented an *Algebra of Communicating Processes (ACP)*. ACP provides again a process model similar to CCS and CSP (and is therefore not discussed here in detail), however we remark that its presentation tries to emphasise the algebraic aspect of process algebra more than other approaches.

π-calculus. The *π-calculus* of Milner, Parrow, and Walker [96], also described in the books [95, 125], is most prominent and influential among the calculi which introduce some form of mobility into their process model. In pure π-calculus this is mobility of links, which can be passed around and then used for interaction with other processes. This is achieved syntactically by action prefixes $\pi ::= \bar{x}y \mid x(z) \mid \tau$, resembling the value-passing version of its ancestor CCS. However, values and actions are no longer distinct but unified in the concept of *names* and this implies, most importantly, that communicated names can be used as new communication links after reception. Name restriction $\nu z P$ limits the scope of a name to the process P, similar to restriction in CCS, but the scope of a restriction may change dynamically as the result of communication (*scope extrusion*). Replication $!P$ is the means to express infinite behaviour via the equation $!P \equiv P \mid !P$ of the structural congruence.

The use of structural congruence hints at the fact that the operational semantics of π-calculus is frequently given in CHAM style, leading to very concise descriptions. The most intricate case of the reduction relation is interaction:

$$(\bar{x}y.P_1 + M_1) \mid (x(z).P_2 + M_2) \rightarrow P_1 \mid P_2[y/z]$$

Here, M ranges over the subset of processes with only prefixing, choice, and termination. For behavioural reasoning, bisimilarity cannot be readily applied in CHAM style as it lacks observability through labelling. Instead, observability predicates (*barbs*) have to be defined and used in related notions of equivalences. In the case of π-calculus, the barb is a predicate $P \downarrow \beta$, meaning that P can perform a communication on the name β. Alternatively, the definition of auxiliary predicates can be circumvented by defining an equivalent labelled transition semantics.

2.1.2 Broadcast Calculi

The majority of process calculi take point-to-point communication as primitive, a natural choice considering the large number of systems in which only two agents will interact at any time. However, it is just as easy to name systems which require or are easier to implement with broadcast: mobile telephony, Ethernet, and, in our case, wireless networks. In order to model these systems, it seems that a dedicated calculus with broadcast as primitive could greatly simplify matters, since complicated encodings of broadcast in point-to-point fashion can be avoided. Moreover, Ene and Muntean [48] have shown a separation result between broadcast-based and point-to-point calculi, namely that no *uniform* encoding of broadcast calculi into their point-to-point ancestors exists (bπ-calculus/π-calculus, CBS/CCS). While this does not prove that there is no encoding at all, it supports the view that the encoding may be too unwieldy to be useful.

CBS. Prasad [120, 121, 122] has introduced the *Calculus of Broadcasting Systems (CBS)* as the first calculus with broadcast as communication primitive. It is inspired by local area networks in which nodes overhear all messages, but can filter addressed messages, and by the intuitive example of everyday speech, where speaking is autonomous but hearing is dependent on it. The design of CBS closely follows that of CCS and inherits its static model.

As in value-passing CCS, there are expressions e, values v, and variables x. The communication constructs are $x?P$ and $e!P$ and describe listening and transmission of an expression e, respectively. Transmission is performed autonomously while reception is a response to transmission and only performed for listening processes; otherwise the transmitted value is lost. In the operational semantics, this is achieved by a labelled transition relation with structured labels: they show both the expression being transmitted and whether the expression is transmitted, received, or lost by the respective process, using the symbols $\{!, ?, :\}$. As an example, note the labels in the following transition rules for a transmitting and a receiving process:

$$e!P \xrightarrow{v!} P, \text{ if } e \text{ evaluates to } v \qquad\qquad x?P \xrightarrow{v?} P[v/x]$$

The rule for parallel composition ensures that all listening processes receive a transmitted value and only one value can be transmitted at a time, avoiding clashes:

$$\frac{P_1 \xrightarrow{v\sharp_1} P_1' \qquad P_2 \xrightarrow{v\sharp_2} P_2'}{P_1 \mid P_2 \xrightarrow{v(\sharp_1 \circ \sharp_2)} P_1' \mid P_2'}$$

An algebra for the composition $\circ$ of $\sharp_1, \sharp_2 \in \{!, ?, :\}$ realises this semantics. Behavioural relations for CBS are given in [66] and include a strong and a weak version of bisimu-

lation. There is however little work which puts these relations to good use, and none of the initially envisioned scenarios such as local area networks have actually been modelled.

Ostrovsky, Prasad, and Taha [111] have proposed *Higher-Order Calculus of Broadcasting Systems (HOBS)*, a higher-order version of CBS so that messages are processes instead of values. HOBS also permits to model asynchronous communication via an explicit buffer statement. Besides, it offers a variety of highly specialised language constructs to model Ethernet-style communication. While HOBS seems to realise this particular application, it is still a long way from a general model for higher-order broadcast communication.

In [123], Prasad proposes *Mobile Broadcasting Systems (MBS)* which merge the broadcast paradigm with the idea of bounded places for computation from Mobile Ambients (see §2.1.3). Processes can transmit in located rooms, which can however not be nested, and walk silently in between rooms. As a work in early stage, a striking application of this concept remains to be found. It does not seem to simplify the modelling of wireless computing, since processes cannot be in two rooms at the same time.

bπ-calculus. Ene and Muntean [47] and the dissertation of Ene [46] describe *bπ-calculus*, a π-calculus variant which has broadcast as main primitive, building on the ideas of CBS. In contrast to CBS, it uses channels so that broadcast is transformed into multicast: all processes listening on a channel a, written $a(x).P$, receive messages put on this channel by $\bar{a}x.P$:

$$a(x).P \xrightarrow{a(z)} P[z/x] \qquad\qquad \bar{a}x.P \xrightarrow{\bar{a}x} P$$

A group of inference rules like the following example realises this behaviour analogously to CBS:

$$\frac{P_1 \xrightarrow{\bar{a}x} P_1' \qquad P_2 \xrightarrow{a(x)} P_2'}{P_1 \mid P_2 \xrightarrow{\bar{a}x} P_1' \mid P_2'}$$

Behavioural equivalences are developed like in π-calculus as a mix of barbed and labelled bisimulations. As in the case of CBS, only few elementary programming examples are provided but no modelling of prototypical broadcast systems has been attempted.

2.1.3 Calculi with Distribution

According to our overview in the introduction to §2.1, process calculi can be enriched with locations to describe distribution of processes. Two lines of research can be distinguished [31]. Firstly, locations can be used to differentiate parallel components in process calculi with non-interleaving semantics. A semantics is called *interleaving* if

it contains an *expansion theorem* stating that parallel composition can be equivalently expressed by choice and sequential composition, e.g. $a.0 \mid b.0 = a.b.0 + b.a.0$. This is in accordance with the view that an observer perceives simultaneous actions as occurring sequentially in some order. Otherwise a semantics is called *non-interleaving* or "true-concurrency". Secondly, locations have been used to describe distributed mobile computation. We will be concerned with this latter view here.

Distributed Variants of π-calculus. There are many distributed variants of π-calculus: the *Dπ-calculus* by Hennessy and Riely [67], the *dpi calculus* by Sewell [127], *Nomadic Pict* by Wojciechowski and Sewell [135]; others, such as the *Distributed Join Calculus* [52] by Fournet et al., are at least heavily influenced by (asynchronous) π-calculus. All these approaches have in common that they focus on modelling process mobility, with applications such as resource access control; many develop type systems for general distributed mobile agents or to support an analysis of a particular application. As such, they seem only weakly related with our approach and so we do not discuss them in detail.

Mobile Ambients. We will however briefly review *Mobile Ambients* of Cardelli and Gordon [30], which have been applied to similar goals but provide a rather novel and general model not relying on π-calculus. The Mobile Ambients calculus emphasises process mobility over communication. Ambients $n[P]$ are located places for parallel computation which can be nested. Primitive actions are movement of ambients in and out of other ambients, **in** n | **out** n | **open** n. A typical rule is:

$$n[\textbf{in}\ m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$$

Communication primitives are given as well in π-calculus style, however they are not necessary to make the calculus Turing-complete. The initial motivation for Mobile Ambients comes from mobile agents and firewall crossing. To reason about these, contextual equivalence is developed, stating that two processes are equivalent if they admit the same observations inside an arbitrary term context.

Klaim. Bettini et al. [14] have developed *Klaim*, a language which owes concepts to both process algebra and coordination languages. In the realm of *coordination languages*, the view is held that programming languages can be built out of two ingredients [54]: a computation model and a coordination model. A coordination language such as *Linda* [53] can then be added to any base language, which itself is sequential, to achieve a full language with communication and concurrency. At the centre of Linda's communication model are so-called *tuple spaces*, which are multisets of tuples (messages composed of multiple fields). Tuples can be put and retrieved again via pattern

matching mechanisms, realising asynchronous communication.

Klaim introduces a distributed variant of the Linda paradigm by having multiple tuple spaces which can be accessed remotely. Technically, they introduce a new syntactic category of *nets* N which contain both located processes P and tuples t:

$$N ::= l ::_\rho P \mid l :: \langle t \rangle \mid N_1 \parallel N_2$$

Here, ρ is a so-called *allocation environment* mapping locality variables to localities in P. Locality variables and the allocation environment permit writing programs for distributed environments while ignoring their precise allocations. Linda actions such as **out**(t) and **in**(t) allow on the one hand that tuples are put and retrieved from locations, on the other hand also higher-order communication, to model code mobility and remote execution. There is no relations-based behavioural theory for Klaim. Instead, the focus has been to use program analysis methods like type systems to derive properties of programs.

2.1.4 Calculi for the Analysis of Security Protocols

With their focus on communication as modelling primitive for concurrent systems, process calculi seem to be especially suitable for the analysis of protocols, which themselves deal with message exchanges in networks. Especially security protocols, which use cryptography to establish security properties like authentication or confidentiality between communication partners, have been in the focus of research. In order to express these protocols, calculi have to provide ways to model cryptographic primitives.

CSP. One of the first applications of process calculi in this setting is due to Lowe [86], where CSP (see §2.1.1) was used to find a flaw in the Needham-Schroeder Public Key Protocol. Encryption of a message m with a key k is expressed by a CSP event $Encrypt.m.k$, but no semantic precautions are made that processes handle this event according to the perfect cryptography assumption (for an explanation of this term, see the definition in §5.1). This has to be assured by a proper modelling of the attacker in Dolev-Yao style [44]. The basic approach has been further developed [124] and successfully applied to a variety of protocols.

Spi Calculus. Abadi and Gordon [4] have proposed the *Spi Calculus* as a variant of the π-calculus with explicit modelling of cryptographic primitives. For example, in the case of symmetric key cryptography they extend the set of terms of π-calculus with $\{M\}_N$, representing encryption of the message M under key N, where M and N are names. Process syntax contains the new statement **case** L **of** $\{x\}_N$ **in** P and the reduction relation $>$ is used to specify that only encryptions bearing the correct key N

can be decrypted:

$$\textbf{case } \{M\}_N \textbf{ of } \{x\}_N \textbf{ in } P \ > \ P[M/x]$$

Asymmetric encryption can be modelled similarly. They derive the security properties authentication and secrecy for systems *sys* with an obviously secure specification *spec*, by proving the equation $sys \simeq spec$. Here, the relation $\simeq$ is testing equivalence, due to [42], where a test is any closed process R, and a process P passes the test if $(P \mid R) \Downarrow \beta$, for a convergence predicate $\Downarrow$. Having the perfect cryptography assumption hardcoded in the operational semantics allows thus for an elegant formalisation of the attacker as an arbitrary process and distinguishes itself from the usual Dolev-Yao approach.

Applied pi calculus. Abadi and Fournet [3] have proposed *applied pi calculus*, a π-calculus variant in which messages can be compound values instead of just names, using primitive functions and equations on terms. The perfect cryptography assumption is ensured by a careful definition of the equational theory for functions representing cryptographic primitives. While losing some of the purity of the π-calculus, they obtain a robust framework in which to express a variety of cryptographic operations without tedious encodings. They present an observational equivalence and two versions of labelled bisimilarity, which turn out to provide the same notion of equivalence. Using these equivalences they can state useful properties of protocols which require elaborate term structures, for example protocols for XML web services [15].

LySa. Bodei et al. [20] introduced the calculus *LySa*, a polyadic version of π-calculus with cryptographic primitives as in Spi Calculus. They use pattern matching as in coordination languages to distinguish on input between messages of different format. They omit to give any notion of behavioural relations, but instead use the calculus as a description language to be exploited by an associated static analysis framework. Buchholtz, Nielson, and Nielson [27] have further developed this calculus to $LySa^{NS}$, which has a more complex view of pattern matching and terms, possibly more convenient for advanced protocol descriptions.

2.2 Protocol Analysis Techniques

A *protocol* defines a sequence of interactions between entities designed to achieve a certain goal. The complexity of protocols and their potentially hostile environment, as well as the subtlety of the properties to be ensured, make protocol design and analysis a difficult domain in which human intuition and informal argumentation have failed many times. Formal analysis techniques are therefore needed to establish a mathematically rigorous connection between protocol modelling and desired goals.

In general, protocol analysis comprises two tasks: finding flaws in protocols that are not correct (*falsification*), or proving their correctness (*verification*). Protocol analysis techniques sometimes provide support for only one of these activities: in general, automated techniques have to focus on falsification, especially in a security context where the undecidability of the general security problem is known [50, 64, 45]; on the other hand, proof-based methods for verification may sometimes only be able to indirectly indicate flaws. In addition, the techniques can also be classified according to the properties they can derive. For trace-based approaches, the property classes for expressing *safety* ("nothing bad happens") or *liveness* ("something good eventually happens") [84] are important, and the following result holds: every property which is formalisable as a set of traces can be written as the intersection of a safety and a liveness property [8]. However, not all properties can be characterised by traces, for example information flow security [90].

Protocol analysis techniques are further distinguished by the protocol classes they can be applied to. Two protocol classes are relevant for our discussion of related work: security protocols and communication protocols. The goal of a *security protocol* is to establish a security property between communication partners, e.g. authentication or secrecy. The protocols usually make use of cryptographic mechanisms such as encryption and digital signatures in order to execute correctly in hostile environments where an attacker may deliberately try to compromise the protocol. A main challenge for analysis techniques in this setting is to accurately model a suitable attacker. For automated analyses, limiting the state space resulting from the attacker is a major problem, because the attacker has infinitely many choices of messages to send at any point during the protocol run. *Communication protocols* on the other hand are designed to establish communication between agents in a network, which includes various tasks such as link establishment, routing, or end-to-end transfer of data. In comparison to security protocols, communication protocols are usually far more complex in terms of message contents and required internal computation. On the other hand, for protocols developed for non-hostile environments it suffices to check functional properties only, and the attacker model can be spared. This simplification does not apply to the more realistic protocols we consider in this dissertation, which try to secure the provided network services via cryptographic mechanisms. In the following, we present approaches targeted towards the analysis of either of these protocol classes.

2.2.1 Analysis of Security Protocols

The analysis techniques described in this section can roughly be classified into proof-based and language-based approaches. *Proof-based* approaches focus on translating protocol steps directly into some mathematical formalism enriched with proof techniques for security analysis. Belief logics, Strand Spaces, and theorem proving will be

described as examples. *Language-based* approaches on the other hand have in common that they start out by modelling a security protocol in some process calculus. Then they exploit standard static analysis techniques such as model checking, control flow analysis, type systems, or abstract interpretation, which have to be adapted in order to yield security results. This latter approach comes with the advantage that tool support is oftentimes directly available.

In the following, we first describe these various approaches, and conclude by giving a critical comparison with our work.

Belief Logics. Burrows, Abadi, and Needham [28] have developed *BAN Logic*, one of the first attempts to reason about security protocols. They introduce statements to express the beliefs of agents about a communication together with an inference system which shows how beliefs can evolve. The central statement is A **believes** X, meaning "A is entitled to believe X". Other constructs include the following: A **sees** X expresses that A has received X at some point; $A \overset{K}{\leftrightarrow} B$ denotes sharing of key K between A and B; $\{X\}_K$ denotes encryption of X under K. The following rule uses these constructs to express that if an agent A receives a message, encrypted under a key known by him, he can also see the decrypted contents:

$$\frac{A \textbf{ believes } A \overset{K}{\leftrightarrow} B \qquad A \textbf{ sees } \{X\}_K}{A \textbf{ sees } X}$$

Using such rules, the main idea is to derive the protocol goals from initial assumptions and the protocol steps. Whilst BAN Logic has been seminal to security analysis, it is also famous for "proving correct" the original Needham-Schroeder Public-Key protocol, which has later been found flawed by Lowe [86]. It has since then been controversially discussed whether this constitutes a flaw of the logic or a misunderstanding of the protocol goals. While we do not want to add to this discussion, we believe that it highlights the fact that interpretations of BAN analyses can be extremely subtle, which may be seen as a weakness of the framework. It is also important to note that the framework is limited in its applicability in that it can handle authentication only. Extensions to the logic which address some of these defects have been addressed in [56, 89, 130].

Strand Spaces. Fabrega, Herzog, and Guttman [51] have developed the *Strand Spaces* approach. A strand space is a set of *strands*, which are sequences of events a single protocol participant may engage in. Events are transmission and reception of messages, expressed by the signed terms $+t$ and $-t$, respectively. The space is partially ordered via two relations on signed terms: relation $\longrightarrow$ represents causality induced by message sending and reception, i.e. $+t \longrightarrow -u$ holds iff $t = u$; relation $\Longrightarrow$ expresses that events are successive on a strand, for example one participant may emit t_1 and expect to re-

ceive t_2 in its next interaction, $+t_1 \Longrightarrow -t_2$. Reasoning takes place in *bundles*, which are causally connected sets of strands, and in particular one is interested in all bundles which correspond to the strand expected by a certain protocol participant. These bundles may also contain strands of the Dolev-Yao attacker, and may be seen as all possible combinations of events consistent with the protocol participants' view of the protocol run. Using mainly inductive reasoning techniques, various security properties can be proved. While a failed proof might suggest a particular flaw, the main contribution of Strand Spaces can be seen in providing a framework for proving protocols correct and giving new insights on their workings. Song, Berezin, and Perrig [129] have provided a mixed model checker and theorem prover called Athena which is based on the Strand Spaces model.

Theorem Proving. Paulson [113] employs Isabelle/HOL, an interactive theorem prover for Higher-Order Logic. Protocols are expressed via inductive definitions to produce sets of traces, similar to the CSP approach described below, and authentication and secrecy properties are specified and proved as predicates over these traces. Infinite state systems can be handled in this way, however extensive user interaction might be required for generating the proofs.

Model Checking. In *model checking*, correctness analyses are performed automatically by enumerating all possible states of a system and exhaustive exploration of the resulting state space. Correctness requirements can be specified either as formulae in some temporal logic (assertion checking), or as abstract system specifications (refinement checking). The main practical challenge of model checking is to overcome the so-called *state explosion problem*: the size of the state space grows exponentially with the size of the system description. A general verification technique, model checking has been applied to protocol analysis, and we mention two lines of work here.

Lowe [86] has used CSP (see §2.1.1) and the *Failures Divergences Refinement (FDR) Checker* to establish results on security protocols. FDR has two CSP processes as input: an "implementation" of the protocol, which includes the modelling of the attacker, and a "specification", producing only the traces of correct protocol runs. If the implementation is found to refine the specification, this means that it could actually replace the specification and is thus correct. The approach is very general, has very good tool support, and has discovered many protocol flaws, as documented in the book by Ryan and Schneider [124]. As a downside, resulting from the underlying finite state model checking methodology, it is sometimes not possible to analyse protocols in all generality and finitary restrictions or drastic simplifications have to be made.

Basin, Mödersheim, and Viganò [10] have used state-of-the-art model checking techniques to overcome some of these restrictions and to handle very large, or even infinite, state spaces. They employ the *lazy infinite-state approach*, which encodes an infinite

tree as an element of a data type in a lazy programming language, leading to a finite representation of the model which can be extracted on-demand. On the other hand, they represent terms symbolically to avoid enumerating the possible messages the Dolev-Yao attacker can generate with the *lazy intruder* approach.

Control Flow Analysis. *Program analysis* is concerned with static techniques to predict the behaviour of programs when executing on a computer [104]. Bodei et al. [23, 24, 20, 22] have shown that such techniques, namely *control flow analysis*, can also be applied to process calculi, and have derived secrecy and authentication properties in variants of the π- and Spi Calculus. In [23], a control flow analysis for π-calculus is presented which focuses on the usage of channels. The analysis result overapproximates the set of channels a binder may be bound to, and the set of channels that may be sent along a given channel. The analysis is then used to establish a simple confinement property, namely that a certain set of channels will never be communicated. In [24], the approach was shown to scale to a calculus with perfect cryptography, a variant of Spi Calculus. Also, the Dolev-Yao notion of security turns out to be expressible in a control flow analysis framework, and can be used to show a non-interference property similar to [1]. In [20, 22], the approach is applied to a hand-tailored calculus LySa (see §2.1.4), especially suitable to model a variety of security protocols. The analysis is extended with assertions for origin and destination which are inserted by hand to clarify the intentions of the protocols. With this approach mainly authentication properties are validated, but also secrecy properties can be expressed. The approach is complemented with an implementation using the Succinct Solver, worked out in the dissertation of Buchholtz [26], and has been applied successfully to a variety of protocols to find well-known as well as new flaws [21].

Type Systems. Another program analysis approach to protocol security is based on *type systems*. Abadi [1] presents a typing system for controlling information flow in a polyadic version of the Spi Calculus (see §2.1.4). Data is classified into *Public*, *Secret*, and *Any* (subsuming both other levels), and the goal of the typing system is to establish that if a process P typechecks, it does not leak values of level *Any*. The two main judgements state that a term M is of type T, $E \vdash M : T$, and that process P typechecks, $E \vdash P : OK$. Here, E is an environment consisting of variables and names and their associated security level. For example, the rule for decryption under a secret key looks as follows, where $T \in \{Public, Secret\}$:

$$\frac{E \vdash L : T \qquad E \vdash N : Secret \qquad E, x_1 : Secret, x_2 : Any, x_3 : Public, x_4 : Any \vdash P : OK}{E \vdash \textbf{case } L \textbf{ of } \{x_1, x_2, x_3, x_4\}_N \textbf{ in } P \ : OK}$$

Thus, encryption under a secret key requires a particular format for the contents, for example the last component is a so-called confounder, a fresh value which guarantees that the resulting ciphertext is different from all previous ones. The main theorem of this approach says that if *Any* and *Public* are the only levels in some environment E and P typechecks, then $P\sigma \simeq P\sigma'$ for variable substitutions σ and σ'. Here, the relation $\simeq$ denotes testing equivalence as in the original work on Spi Calculus, and means that an observer cannot distinguish the difference in the values for the variables.

A multitude of type systems for process calculi with respect to various security properties have followed this approach, e.g. [58, 57, 71]. Abadi and Blanchet [2] have directly extended it to a language resembling the applied pi calculus (see §2.1.4), and have shown its relation to the Prolog-based protocol checker ProVerif [19], which translates a process into logic-programming rules which can be automatically evaluated. The logic programming approach itself is rooted in work by Kemmerer, Meadows, and Millen [81], e.g. on the special-purpose evaluation tool *NRL Protocol Analyser* [91].

Abstract Interpretation. *Abstract interpretation* [39] is the accepted generic framework for justifying program analyses. In this framework, analyses are formalised as safe approximations to their exact meaning via the notion of a pair of abstraction and concretisation functions which form a Galois connection. Besides its significance for reasoning about program analysis, it can also be used to construct specific analyses. While there are many such for general security questions, Monniaux [97] has provided one of the few applications to protocol analysis. Here, tree automata are used as abstractions and applied to small protocols with a finite number of runs.

Discussion. In this dissertation, we present a protocol analysis framework for security properties based on a process calculus as modelling language. The works of Buchholtz et al. [26, 22] and Blanchet et al. [19, 2], which we have described above, are based on the same general idea and deserve thus a critical comparison with our approach.

Buchholtz et al. define a process calculus LySa, which we have described above as polyadic version of the π-calculus with cryptographic primitives. The main difference to the calculus CBS$^\sharp$ we develop in Chapter 3 is thus the point-to-point (as opposed to broadcast) communication model. Furthermore, the messages of CBS$^\sharp$ are terms of a free term algebra with functions, whereas they are tuples in the case of LySa. However, CBS$^\sharp$ adopts a notion of prefix matching from LySa, by rephrasing tuple matching as function matching. In contrast to LySa, CBS$^\sharp$ is equipped with locations to express distribution, and a notion of a persistent store at every location as necessary prerequisites for reasoning about routing protocols. Cryptographic primitives are hardcoded in LySa and thus Spi Calculus-like reasoning about attackers as arbitrary processes is possible, which is not the case for CBS$^\sharp$ where an attacker has to be modelled carefully to preserve the perfect cryptography assumption. No behavioural relations are given

for LySa, while we use mediated equivalence (see §3.3) to define security properties. Prompted by the different application areas of the two calculi (security protocols for authentication and confidentiality for LySa, routing protocols for wireless networks for CBS$^\sharp$), the goals of the analyses performed on them are rather different as well: Buchholtz et al. analyse assertions about the origin and destination of messages together with a reference monitor semantics (which aborts if a message is decrypted at an unintended destination); CBS$^\sharp$ uses a model of the routing tables together with a notion of the network topology to reason about routing paths in a network. The security analysis for LySa uses a standard Dolev-Yao attacker which is expressed by additional control flow analysis rules, whereas we use a modified Dolev-Yao model of multiple attackers which do not necessarily have control of the network and can be directly modelled as CBS$^\sharp$ terms. The analyses of both approaches are defined using control flow analysis, and the encouraging results obtained by Buchholtz et al. concerning the expressivity of the constraint language have prompted us to choose the Succinct Solver as a constraint solving tool as well. Besides the different analysis goals, one can also observe differences in the precision and complexity of both analyses (with our analysis being more precise, yet also more complex), which are further outlined in §4.5.5.

Blanchet et al. use a derivative of the applied pi calculus to describe protocols. This calculus replaces the equational theory of applied pi by an application of destructors, and is thus close to our handling of terms, however CBS$^\sharp$ has a prefix matching construct on function components in addition to that. Again, CBS$^\sharp$ has distribution and a notion of persistent memory in addition to these features. Blanchet et al. define a type system for their calculus with the goal that well-typed processes do not reveal their secrets. This type system is compared against the automatic protocol checker ProVerif, which is based on Prolog, and soundness of the checker with respect to the type system is proved. Protocol processes can be automatically translated into Prolog rules usable by the checker. In contrast, our approach is based on behavioural equivalences and control flow analysis, which does however have some similarities from a general perspective: the behavioural equivalences play a similar role for the control flow analysis as the type system for ProVerif, namely to express the desired security property on the language level; also, we implement the control flow analysis using a solver for a fragment of first order logic, thus comparable to the Prolog checker implementation. For the employed attacker model, similar comments as in the case of Buchholtz et al. apply.

2.2.2 Analysis of Communication Protocols

We focus on discussing work concerned with routing protocols rather than general communication protocols. In this area, the main interest has been on analysing general correctness properties, but some more recent work addresses security properties as well.

Model Checking. Bhargavan, Obradovic, and Gunter [17] and Obradovic in his dissertation [110] have studied verification of routing protocols, with the Internet routing protocol RIP and AODV (see §1.1.3) as examples. For a loop-freedom property expressed in temporal logic the model checker SPIN finds counterexamples, some of which have not been discovered by manual inspection and led to an improvement of the corresponding IETF standards. They complement the approach with theorem proving in Higher-Order Logic in order to show that their amendment of AODV is indeed correct.

Also Wibling, Parrow, and Pears [133] have worked on model checking of protocol specifications, with LUNAR (a lightweight variant of AODV) as an example. They employ SPIN to check the correctness of message delivery, and they validate timing properties using UPPAAL, a model checker for real-time aspects. They apply their analysis to all fixed topologies of a maximum of five nodes, but experience state-space explosion. When trying to add mobility, they equally have to conclude that exhaustive search is "no longer feasible" and focus on interesting hand-made scenarios.

Chiyangwa and Kwiatkowska [34] have pursued a similar line of research and use UPPAAL to check timing properties on a fairly complete model of AODV. They use a fixed linear topology with 14 nodes. In order to achieve this, 12 intermediate nodes are simulated using one super node which is looping 12 times. Using their findings, they criticise the design of AODV for assuming a constant network size which limits the number of hops a message can be maximally forwarded along. They suggest an improvement where nodes can learn interactively about the actual size.

In the work of Zakiuddin et al. [140] CSP and FDR (see §2.2.1) have been applied to a model of the Cluster-Based Routing Protocol (CBRP) for a self-configuration (leader-election) problem. Their approach is interesting because they are able to describe all topologies and topology changes, for a small finite number of nodes. They succeed in integrating the mobility aspect by modelling links as individual processes which can be either up or down.

Das and Dill [41] use model checking in conjunction with predicate abstraction to prove loop-freedom in AODV. *Predicate abstraction* is an efficient way to reduce infinite state systems to tractable finite state systems: the concrete states of a system are mapped to abstract states according to their valuation under a finite set of predicates. The abstraction is conservative in the sense that any property proved on the abstract systems holds on the concrete one. Their method yields a general proof on a simplified version of AODV for which the abstraction predicates can be automatically generated. In general, finding the predicates might require human intervention.

Conformance Testing. The approach of Bhargavan et al. [16] changes the focus from proving properties about specifications to validating a particular implementation of a specification. Their idea is to use the fact that most performance evaluation of communication protocols is done using simulators, e.g. the network simulator ns-2, and

use the simulator's output traces as input for a logic-based checker Verisim. They propose to use this hybrid system with a "Repair First Bug" strategy (i.e. repair the first bug you find, then analyse again) to improve debugging.

Bishop et al. [18] determine the conformance of network protocol specifications with available real-world implementations. They use a general-purpose proof assistant (HOL) to specify a (large scale) full model of TCP, UDP, and Sockets API. then determine the differences between the specification and execution traces obtained from implementations with a purpose-built symbolic model checker.

Coloured Petri Nets. Xiong, Murata, and Tsai [137] use *Coloured Petri Nets (CPNs)* as a modelling language for AODV. They propose the *topology approximation* technique to model mobility: receiving messages at different nodes is controlled via a function based on the probability of receipt, equal for all nodes. The CPN is then simulated using this technique for a fixed size network of few nodes. The derivation of the receipt probability is based on some non-trivial assumptions such as "every node has the same degree". Together with the fact that the model is run as a simulation it is clear that this approach is not suitable for verification but might be valuable for finding protocol flaws.

In further work Xiong, Murata, and Leigh [136] overcome some of the previous deficiencies by using a tool to generate the state space graph from a Petri Net model and find in this way a routing loop flaw in AODV. However, the critical assumptions of the topology approximation technique remain.

Strand Spaces. Yang and Baras [139] consider an analysis of AODV in the presence of attackers. They use an extension of the Strand Spaces approach (see §2.2.1) and provide tool support similar to the hybrid model checker and theorem prover Athena. They apply their approach to a small network with an explicit intruder and can conclude that it is possible to break an otherwise stable route.

Simulation Paradigm. Buttyán and Vajda [29] and Ács, Buttyán, and Vajda [5] use the *simulation paradigm*, introduced by [11]: one constructs a *real-world model*, i.e. an implementation and a realistic attacker, and an *ideal-world model*, i.e. a specification plus an attacker which describes tolerable imperfections of the system. Both models are described with sets of Turing machines which represent protocol participants. Security of the protocol is proved by showing that the effects of the real-world adversary on the implementation can be simulated by the ideal-world attacker in the specification. a methodology resembling refinement in the CSP/FDR approach. Using the simulation paradigm they can prove correct a secure routing protocol for ad-hoc networks. A weakness of their approach seems to be that no tool support is available.

2.3 Security of Mobile Ad-Hoc Networks

In §1.1 we have introduced mobile ad-hoc networks and have given a general overview of their characteristics and security issues. In this section, we extend this discussion by a review of important protocols, and describe related work on security aspects in this setting.

2.3.1 Background: Fundamental Routing Protocols

Based on the general routing algorithms described in §1.1.2, a multitude of protocols for mobile ad-hoc networks have been developed. We review some of the fundamental proposals.

Proactive Routing. Perkins and Bhagwat [116] have developed *Destination Sequenced Distance Vector (DSDV)* routing, one of the first routing protocols for mobile ad-hoc networks. It is based on the classical distance vector algorithm DBF (see §1.1.2) but makes some improvements to adjust to the new setting. Most importantly, *sequence numbers* are introduced to distinguish stale routes from new ones and thus avoid the formation of loops. Every node maintains an own sequence number which is incremented and sent with every route update, and route table entries for a given destination are labelled with its respective sequence number. If multiple routes are available, the route labelled with the highest, i.e. most recent, sequence number is chosen. If two routes have the same sequence number then the route with the least distance is used. In addition to this technique, DSDV describes heuristics such as event-driven updates to improve efficiency.

Jacquet et al. [76] have proposed *Optimized Link State Routing (OLSR)*, which has also been standardised by the IETF [36]. It optimises classical link state routing for usage in mobile ad-hoc networks. The main idea is to minimise the overhead from flooding of routing information by two techniques: firstly, only selected nodes, called *multipoint relays (MPRs)*, are used to retransmit routing messages; secondly, the amount of link state information which needs to be flooded is reduced to the information about links to a node's *multipoint relay selectors*, i.e. nodes which have selected the node as MPR. By information obtained from periodical transmissions to neighbouring nodes it can be ensured that sets of MPRs and MPR selectors are chosen so that shortest path routes can be provided.

On-demand Routing. We have already described AODV in §1.1.3 as one of the leading approaches to on-demand routing in mobile ad-hoc networks.

Johnson and Maltz [78, 79] propose *Dynamic Source Routing (DSR)* as a variant approach. As a source routing protocol, the sender of a packet prescribes in the header a list of addresses through which the packet is to be forwarded in order to reach the

destination. Basic operation is divided into two steps: route discovery and route maintenance. *Route discovery* is only initiated on-demand, and consists of a *route request* control packet being broadcast by the source and rebroadcast by intermediate destinations. The route is built incrementally by appending addresses to the route request until the destination is reached (*route accumulation*), where it is answered with a *route reply* sent back to the source. *Route maintenance* relies on link layer acknowledgement to detect broken links, which are then reported with a *route error* message to the source, which has to reinitiate route discovery.

Hybrid Approach. Haas [62] has proposed the *Zone Routing Protocol (ZRP)*, which incorporates both proactive and on-demand behaviour. It has been turned into a generic routing framework in later work [63]. The approach is based on the notion of a *routing zone*, defined for each node as the set of nodes within some chosen maximum distance. For routing within the zone, a proactive Intrazone Routing Protocol (IARP) is to be used, and for interconnecting with other zones an on-demand Interzone Routing Protocol (IERP). Intrazone routing can be provided by some link-state protocol, the performance of which is however drastically improved since flooding is cut off after the maximum distance. Interzone routing can be implemented using standard source routing techniques, however with the difference that an underlying *bordercast* paradigm is used, so that only nodes on the borders of a zone rebroadcast a control packet.

2.3.2 Background: Secure Routing Protocols

The protocols described in §2.3.1 are only supposed to execute correctly in non-hostile environments. Considering the general accessibility of the wireless medium, this seems to be a very strong assumption which cannot hold in realistic applications. The works we describe in this section address this issue by providing extensions and modifications to the fundamental algorithms in order to ensure security.

Papadimitratos and Haas [112] have developed the *Secure Routing Protocol (SRP)*, a secure protocol extension scheme for source routing algorithms such as DSR. The protocol requires shared keys between each source-destination pair. Integrity of a route request is protected by a Message Authentication Code (MAC) [55] computed by the source using the key shared with the destination. Intermediate nodes cannot verify the MAC but append their addresses to the request and rebroadcast it, according to the underlying source routing protocol. The destination verifies the MAC and, if found correct, sends back a route reply protected with a MAC computed with the shared key. Intermediate nodes forward the reply along the route without taking any other actions. If the source receives the reply and can correctly verify the MAC, the route is accepted.

Guerrero Zapata and Asokan [61] have introduced *Secure Ad-hoc On-Demand Distance Vector (SAODV)* routing, which has also been submitted to the IETF as an

Internet Draft [59]. The protocol provides extension headers for AODV which rely on two authentication mechanisms: digital signatures and a variant of hash chains, in order to protect non-mutable and mutable data in routing messages, respectively. Both source and destination sign non-mutable data in a routing message, and intermediate nodes verify the signatures before updating their routing tables. To protect mutable distance information, i.e. the number of hops a message has travelled already, a hash function is applied repeatedly at every hop to an initial seed.

Ariadne, a stand-alone secure routing protocol for mobile ad-hoc networks, has been proposed by Hu, Perrig, and Johnson [74]. It comes in three variants corresponding to underlying techniques for data authentication: digital signatures, MACs, and TESLA, which is an efficient authentication scheme based however on the somewhat unrealistic requirement of clock synchronisation. Ariadne is based on a source routing algorithm, hence best comparable to SRP. As a difference, intermediate nodes use per-hop hashing and signatures to prevent removal of nodes from the accumulated route.

Hu, Johnson, and Perrig [73] have also developed *Secure Efficient Ad Hoc Distance Vector Routing (SEAD)*, which is based on the distance vector routing protocol DSDV. SEAD deals with the limited scenario of attacks that try to modify entries in routing tables by forged update messages. For this, hash chains are used, where it is assumed that authenticated elements of the hash chain can be distributed before the start of the protocol. They describe a system of choosing elements of hash chains to authenticate sequence number and distance information for each routing table item to be updated.

Sanzgiri et al. [126] have described *Authenticated Routing for Ad hoc Networks (ARAN)*, a protocol which makes use of cryptographic certificates and digital signatures. It is thus assumed that a trusted certificate server exists. The basic protocol is distance vector routing in the fashion of AODV. However, when sending out routing information, nodes append their certificates and sign messages with their private key. Receiving nodes examine the certificates and verify the signatures, which are subsequently removed and replaced with their own certificate and signature if they are issued by intermediate nodes. Nonces and timestamps are used to prevent replay attacks.

2.3.3 Security Properties

Various security protocols, as described in §2.2, have been subject to much debate about the subtleties of which security property will be ensured by its execution. This is no different in the area of communication protocols as the discussion of the following works shows.

Hu, Perrig, and Johnson [74] characterise the security offered by their protocol Ariadne as follows: if there exists an uncompromised neighbour of the destination on a shortest path between itself and the source which does not contain any compromised

nodes, this path will be returned in a route reply. The idea is that Ariadne ensures that at least one working route is returned. The property is informally stated and argued for.

Papadimitratos and Haas [112] try a protocol correctness proof using BAN Logic, to verify their claim that "SRP guarantees acquisition of correct topological information". As described in §2.2.1, BAN Logic is a technique developed for authentication properties only, it is thus evident that the proposed "proof" is flawed. An attack on SRP has been subsequently described by Buttyán and Vajda [29] who also restate the property of Papadimitratos and Haas for general routing protocols.

Guerrero Zapata and Asokan [61] consider a new property *import authorisation*, which they state informally. Import authorisation says that if a routing update is received at some intermediate node, the affected node (i.e. the source or destination node) should have the (remote) authority over execution of this update. They state that authentication of origin, entity authentication, and integrity are additional properties which are required for establishing import authorisation.

Yang and Baras [139] formulate a safety and a liveness condition for secure routing. The safety condition asserts that discovered routes are stable. The liveness condition says that if a route exists it can be found. Both properties are formally stated in temporal logic.

2.3.4 Attacker Models

Needham and Schroeder [101] have introduced the view that in order to prove an authentication protocol correct, a powerful attacker which controls all communication paths has to be considered. This has become the prevalent view for the analysis of any security protocol and one might even be tempted to assume that is could be the right model for *all* kinds of protocols. However, the works described in the following directly address the question of the alternate attacker models.

Hu, Perrig, and Johnson [74] devise a hierarchy for attackers in mobile ad-hoc networks. *Passive* attackers can only eavesdrop and are not considered a major threat. *Active* attackers on the other hand can also inject messages, and can be characterised by the number of compromised nodes n and total number of nodes m owned; the notation *Active-n-m* is introduced for this attacker. The idea is that compromised nodes were honest protocol participants once, however they have been captured or their cryptographic information has been revealed to the attacker in some other way. It is further assumed that the attacker distributes the collected cryptographic information to all the nodes he owns.

Creese et al. [40] have thought about restraining the classical Needham-Schroeder attacker for various applications in ubiquitous computing. The applications they consider suggest that it might be helpful to consider a twin-channel threat model, where

possible actions of the attacker are separated into those on a low security channel (e.g. the wireless medium) and a more reliable one (by direct physical interaction of the user with some device, e.g. reading information off a display). While this model does not seem to apply to ad-hoc networks, it highlights the fact that applications in ubiquitous computing might require attacker models different from the Needham-Schroeder approach.

Chapter 3

The Calculus CBS$^\sharp$

In this chapter we present CBS$^\sharp$, a process calculus for modelling and reasoning about mobile ad-hoc networks. This application domain calls for several distinctive modelling features: most importantly, we inherit broadcast as the base communication paradigm from the Calculus of Broadcasting Systems (CBS) [122] (see §2.1.2), with the difference that sent messages are not received globally, but only by adjacent neighbours of the sending node, a mechanism which we call *local broadcast*. The notion of adjacency is made explicit by the concept of *connectivity graphs* and, more general, *network topologies*, which effectively separate process actions from process connectivity. This separation is essential for modelling mobile wireless behaviour since the possibility of a connection is determined by environment conditions such as node movement, but never by process actions. Another characteristic of CBS$^\sharp$ is its explicit notion of a computational entity as a located pair of process and store. Having explicit stores proves to be more than a modelling convenience because behavioural equivalences and security reasoning rely heavily on the notion of storage. In addition, processes of CBS$^\sharp$ communicate terms of a free term algebra with functions and can use a statement for case distinction with pattern matching. This enables us to describe complex data, as required for modelling communication protocols, and to express cryptographic primitives.

This chapter is in four sections. We first present in §3.1 the syntax and informal semantics of CBS$^\sharp$. In §3.2 an operational semantics is provided and the notion of network connectivity that the semantics is based upon is introduced. A notion of behavioural equivalence for CBS$^\sharp$ is developed in §3.3. We conclude the chapter in §3.4 with notes on notation and cryptographic modelling.

3.1 Syntax

In this section, we introduce the language of CBS$^\sharp$. It is defined via three syntactic categories which are presented in Table 3.1: terms, processes, and networks. In ad-

dition, stores are defined as non-empty sets of terms. Networks, as the main entities of the calculus, consist of parallel compositions of located *nodes*, each of which is a pair of a process and a store. A network thus represents a collection of self-contained devices which are running processes that allow them to communicate with other devices or interact with their private memory component. In the following, we informally discuss the meanings of the constructs in each of the syntactic categories and explain the notions of binding and substitution.

Terms		
T ::=	n	names, $n \in \mathcal{N}$
|	x	variables, $x \in \mathcal{X}$
|	$f(\widetilde{T})$	function application, $f \in \mathcal{F}$
Processes		
P ::=	**nil**	termination
|	**out** $T.P$	sending
|	**in** $x.P$	reception
|	**store** $T.P$	storage
|	**read** $x.P$	retrieval
|	**case** T **of** $f(\widetilde{T};\widetilde{x})\ P_1$ **else** P_2	case distinction, $f \in \mathcal{F}$
|	$P_1 \mid P_2$	parallel composition
|	$!P$	replication
Networks		
N ::=	$n[P,S]$	node, $n \in \mathcal{N}_{loc} \subseteq \mathcal{N}$, S non-empty set of terms
|	$N_1 \parallel N_2$	parallel composition

Table 3.1: Syntax of CBS♯

Terms. Let $\mathcal{N}$ denote a countably infinite set of names, $\mathcal{X}$ a countably infinite 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** is inductively defined in Table 3.1. The informal meaning of the individual constructs can be summarised as follows:

(1) *Names* $n \in \mathcal{N}$ are the simplest constituents of terms. They are used to represent any kind of atomic data such as message plaintext, cryptographic keys and nonces, or locations. *Locations* are drawn from a specific subset $\mathcal{N}_{loc} \subseteq \mathcal{N}$ but handled just as ordinary names, because we want to emphasise that they can be communicated as well. Locations can be thought of as names of protocol participants, or IP and MAC addresses.

(2) We have a distinct set of *variables* $x \in \mathcal{X}$ which – in contrast to names – can be *bound* by some process constructs. A variable is called *free* if it is not bound, and a term is said to be *ground* if it has no free variables. Variables can be replaced by ground terms via *substitution*. The notions of binding and substitution are formalised below.

(3) For function symbols $f \in \mathcal{F}$ with $arity(f) = k$, *function application* is written $f(T_1, \ldots, T_k)$, where $T_1, \ldots, T_k \in \mathbf{T}$. Function application is used to algebraically represent data structures like tuples and lists, messages of different formats, or cryptographic primitives. For instance, $\mathsf{Msg}(text, \mathsf{Hash}(key, text))$ represents a message with two components *text* and $\mathsf{Hash}(key, text)$, where the first represents a plaintext and the second represents the Message Authentication Code of *text* computed with $key \in \mathcal{N}$ and hash function Hash.

The approach of allowing functions as term constructors is comparable to that of the applied pi calculus [3] (see §2.1.4). However, terms of the applied pi calculus are interpreted with an associated equational theory whereas we use a free term algebra and distinguish terms by way of the operational semantics (case distinction with pattern matching).

Notation 3.1 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 $|\,.\,| : \mathbf{T}^* \to \mathbb{N}$ to denote its arity, $|\widetilde{T}| = k$. ⊠

Notation 3.2 Figure 3.2 summarises the meta-variables which are used to range over the various syntactic elements. It shows that in examples we may use more intelligible expressions than in formal developments, in order to support intuition. ⊠

	Meta-variables	**Notation used in examples**
Names	$n, m, \ldots$	$ip, text, \ldots$
Variables	$x, y, \ldots$	$key, hop_cnt, \ldots$
Functions	$f, g, \ldots$	$\mathsf{SymEnc}, \mathsf{Hash}, \ldots$
Terms	$T, U, \ldots$	
Processes	$P, Q, \ldots$	$ReceiveMsg, AODV, \ldots$
Stores	S	
Networks	N, M	AODV_Nodes, AttackScenario, ...

Table 3.2: Notation for meta-variables

Processes. The set of processes $\mathbf{P}$ is inductively defined in Table 3.1. Note that in CBS$^\sharp$, processes are not the topmost syntactic category but are encapsulated by nodes. The boundaries every node provides separates the set of process actions into external and internal ones. The execution of external actions (sending and reception)

for communication with other nodes depends on environment conditions expressed on higher syntactic levels. In particular, these actions cannot be used for communication between processes running in parallel at the same node. This type of interaction is however realised indirectly via a private *store* component which processes can access by internal actions of storage and retrieval. We give a brief account of the intended interpretations of processes:

(1) **nil** represents *termination*; this process is inactive.

(2) The *sending* action **out** $T.P$ can send a ground term T and continue as P. The sending mode is *local broadcast*, i.e. all *adjacent* nodes (and only the adjacent ones) may receive the transmission. Adjacency is defined below in Definition 3.4 via the notion of connectivity graphs.

(3) *Reception* **in** $x.P$ expresses readiness to receive a ground term T and then to continue as P with T substituted for x. For instance, process **in** x. **out** x. **nil** can receive T from adjacent nodes, become **out** T. **nil**, rebroadcast T, and terminate.

(4) The *storage* action **store** $T.P$ can put a ground term T into the store S and continue as P.

(5) *Retrieval* of terms from the store S is represented by the action **read** $x.P$ which takes an arbitrary term $T \in S$ then continues as P with T substituted for x. The action does not remove the read term from the store, nor change the store in any other way. Note that for modelling purposes, the nondeterminism of the retrieval is not significant since we are usually interested only in the fact that there *exists* a continuation process with specific properties.

(6) The action for *case distinction* **case** T **of** $f(\widetilde{T};\widetilde{x})$ P_1 **else** P_2 tries pattern matching of a ground term T with the term $f(\widetilde{T};\widetilde{x})$, where all terms in $\widetilde{T}$ are also ground. For the pattern matching to succeed, there has to exist a sequence of terms $\widetilde{U}$ such that $T = f(\widetilde{T},\widetilde{U})$ and $|\widetilde{U}| = |\widetilde{x}|$. On success, the continuation is P_1 with $\widetilde{U}$ substituted for $\widetilde{x}$; on failure, the continuation is P_2. We assume an implicit arity constraint on function symbols occurring in case expressions, namely that $|\widetilde{T}| + |\widetilde{x}| = \mathit{arity}(f)$, to avoid stuckness. Furthermore we forbid processes which reduce to a case expression matching against non-ground terms in $\widetilde{T}$, which can be enforced by requiring processes to be closed (see below). Case distinction is central to exploiting the structure of the terms of CBS$^{\sharp}$. For instance, in order to destruct a term T of list data type we could use the process **case** T **of** Cons$(:x,xs)$ P_1 **else** P_2, where P_1 can process head x and tail xs in case the list is not empty, and P_2 is invoked otherwise for the empty list Nil. If we instead have the process **case** T **of** Cons$(n;xs)$ P_1 **else** P_2, P_1 would only be invoked for lists T which have head n. This kind of pattern matching has been

employed with similar syntax and semantics in Klaim [14] and LySa [22] (see §2.1.3 and §2.1.4); we generalise these approaches by using a single general case distinction construct for terms with arbitrary functions as root.

(7) Processes can be executed in parallel, written as *parallel composition* $P_1 \mid P_2$. P_1 and P_2 proceed independently from each other, however they can interact by storing and retrieving terms from the store which is accessible to both of them. For example, **store** $T.\,P_1 \mid$ **read** $x.\,P_2$ can be used to communicate term T to process P_2. In contrast, sending and reception however are only for external communication, for example the process **in** $x.\,P_2$ *cannot* directly receive the term T in the context **out** $T.\,P_1 \mid$ **in** $x.\,P_2$. A parallel composition of a set of processes P_i with indices from set $\mathcal{I}$ is abbreviated with the notation $|_{i\in\mathcal{I}}\, P_i$.

(8) *Replication* $!P$ is the operator which allows to express infinite behaviours. It produces as many copies of P as are needed by the equation $!P = P \mid !P$. For instance, the process **!in** $x.$ **store** $x.$ **nil** can receive terms repeatedly and accumulate them in the store.

Binding of names or variables is a central concept of most algebraic calculi and this is no different for CBS$^\sharp$. There are three variable-binding process actions:

in $x.\,P$	x is a binder with scope P
read $x.\,P$	x is a binder with scope P
case T **of** $f(\widetilde{T};\widetilde{x})\ P_1$ **else** P_2	$\widetilde{x}$ are binders with scope P_1

As usual, this can be used to define the following two notions: A variable x is called *bound* if it is a binder or lies within the scope of binder x. A variable is called *free* if it is not bound. A process which does not contain any free variables is called *closed*. It is easy to verify that the set of free variables of a term or process can be computed with the function *fv* defined in Table 3.3. For instance,

$$fv(\textbf{read}\ x.\,\textbf{case}\ y\ \textbf{of}\ f(n,x,z;w)\ (\textbf{out}\ w.\,\textbf{nil} \mid \textbf{store}\ g(y).\,\textbf{nil})\ \textbf{else nil}) = \{y,z\}.$$

Note that names in CBS$^\sharp$ cannot be bound by any construct, they are always free.

In order to express that a process can use a term it receives, retrieves, or finds by case distinction, a notion of replacement of variables by ground terms has to be defined. A *substitution* is a function $\theta : \mathcal{X} \to \mathbf{T}$ from variables into ground terms that is the identity except on a finite set. We write $x\theta$ for the application of θ to x, and $[U_1,\ldots,U_k/x_1,\ldots,x_k]$ (or, equivalently, $[U_i/x_i]_{i=1}^k$) denotes the substitution which is the identity except on $x_1,\ldots,x_k$:

$$x[U_1,\ldots,U_k/x_1,\ldots,x_k] = \begin{cases} U_i & \text{if } x = x_i \text{ for some } i \in \{1,\ldots,k\} \\ x & \text{otherwise} \end{cases}$$

$fv(n)$	$=$	$\emptyset$
$fv(x)$	$=$	$\{x\}$
$fv(\widetilde{T})$	$=$	$fv(T_1) \cup \cdots \cup fv(T_k)$, where $\widetilde{T} = T_1, \ldots, T_k$
$fv(f(\widetilde{T}))$	$=$	$fv(\widetilde{T})$
$fv(\mathbf{nil})$	$=$	$\emptyset$
$fv(\mathbf{out}\ T.\, P)$	$=$	$fv(T) \cup fv(P)$
$fv(\mathbf{in}\ x.\, P)$	$=$	$fv(P) \backslash \{x\}$
$fv(\mathbf{store}\ T.\, P)$	$=$	$fv(T) \cup fv(P)$
$fv(\mathbf{read}\ x.\, P)$	$=$	$fv(P) \backslash \{x\}$
$fv(\mathbf{case}\ T\ \mathbf{of}\ f(\widetilde{T}; \widetilde{x})\ P_1\ \mathbf{else}\ P_2)$	$=$	$fv(\widetilde{T}) \cup (fv(P_1) \backslash \{\widetilde{x}\}) \cup fv(P_2)$
$fv(P_1 \mid P_2)$	$=$	$fv(P_1) \cup fv(P_2)$
$fv(!P)$	$=$	$fv(P)$

Table 3.3: Free variables

Application of substitution θ to a process P or term T means to replace each free occurrence of variable x by $x\theta$. Since we replace variables by ground terms only, unintended capture of variables during this process cannot occur and thus we can omit defining α-convertibility of processes. The following serves as an example for application of substitution:

$$(\mathbf{case}\ f(x, y)\ \mathbf{of}\ f(n; z)\ \mathbf{store}\ z.\, \mathbf{nil}\ \mathbf{else}\ \mathbf{out}\ x.\, \mathbf{nil})[n, g(m)/x, y] =$$
$$\mathbf{case}\ f(n, g(m))\ \mathbf{of}\ f(n; z)\ \mathbf{store}\ z.\, \mathbf{nil}\ \mathbf{else}\ \mathbf{out}\ n.\, \mathbf{nil}$$

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

The intended meaning of stores is to represent private memory components of nodes. This means that only the process running at a certain node can access and modify the node's store. This behaviour emphasises the notion of a node as a self-contained device which can only be accessed by the interface it is offering, i.e. sending and reception actions, and is central for security modelling. It is worth to note that stores are monotone in the sense that we define no process actions which can remove stored terms.

Networks. The set of networks $\mathbf{N}$ is inductively defined in Table 3.1. They are the main entities of CBS$^\sharp$ and depend syntactically on processes and stores.

(1) The *node* $n[P, S]$ is a pair of process P and store S at location $n \in \mathcal{N}_{loc}$. Its capabilities are determined by whatever capabilities P has in conjunction with S, however it restricts the external communication capabilities of P by providing a location. For example, assume two nodes $m[\textbf{out}\ T.Q, \varepsilon]$ and $n[\textbf{in}\ x.P, \varepsilon]$. Node m can perform sending without restrictions, however node n may only receive T if it is connected to m (connectivity is formalised below in Definition 3.4). In this way, a node expresses a computational entity of a network which is restrained by connectivity as an environment condition. Note that sometimes a node is referred to by its location name. Also, while the notation $n[\ldots]$ resembles the one used in Mobile Ambients [30] (see §2.1.3), we do not consider nesting of nodes.

(2) *Parallel composition* of nodes is written $N_1 \parallel N_2$. N_1 and N_2 may proceed independently of each other in general, however they synchronise on sending: if N_1 attempts to send a term T, N_2 has to be silent and determine whether or not to receive T according to the current connectivity, and vice versa. As in the case of processes, a parallel composition of a set of networks N_i with indices from set $\mathcal{I}$ may again be abbreviated with the notation $\|_{i \in \mathcal{I}} N_i$.

We define the *vertex function* $V : \mathbf{N} \to \wp(\mathcal{N}_{loc})$ to yield the set of all locations (vertices) in a network:

$$\begin{aligned} V(n[P, S]) &= \{n\} \\ V(N_1 \parallel N_2) &= V(N_1) \cup V(N_2) \end{aligned}$$

3.2 Operational Semantics

In this section, the informal description of the behaviour of networks as given in §3.1 is made precise by an operational semantics. As a prerequisite for modelling of local broadcast, §3.2.1 defines first a notion of connectivity for networks, which is further generalised in §3.2.2. In §3.2.3 we define and explain the rules for three transition relations which together constitute the operational semantics of CBS♯.

3.2.1 Connectivity Graphs

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

Notation 3.3 For a directed graph (V, E), we write $\{m, n\} \in E$ to denote that both (m, n) and (n, m) are contained in E. ⊠

Definition 3.4 (Connectivity Graph) A *connectivity graph* is a directed graph G with $V(G) \subseteq \mathcal{N}_{loc}$. ⊠

Connectivity graphs are used to describe the connectivity of networks at a particular moment in time.

Example 3.5 For $V_0 = \bigcup_{i=1}^{4} n_i \subseteq \mathcal{N}_{loc}$, the graph

$$G = (V_0, \{(n_1, n_2), \{n_2, n_3\}, (n_3, n_1), \{n_3, n_4\}\})$$

is a connectivity graph. It can be used to provide connectivity information for a network $\|_{i \in \{1,\ldots,4\}} n_i[P_i, S_i]$ as illustrated in Figure 3.1, where bidirectional edges are drawn without arrows. ⊠

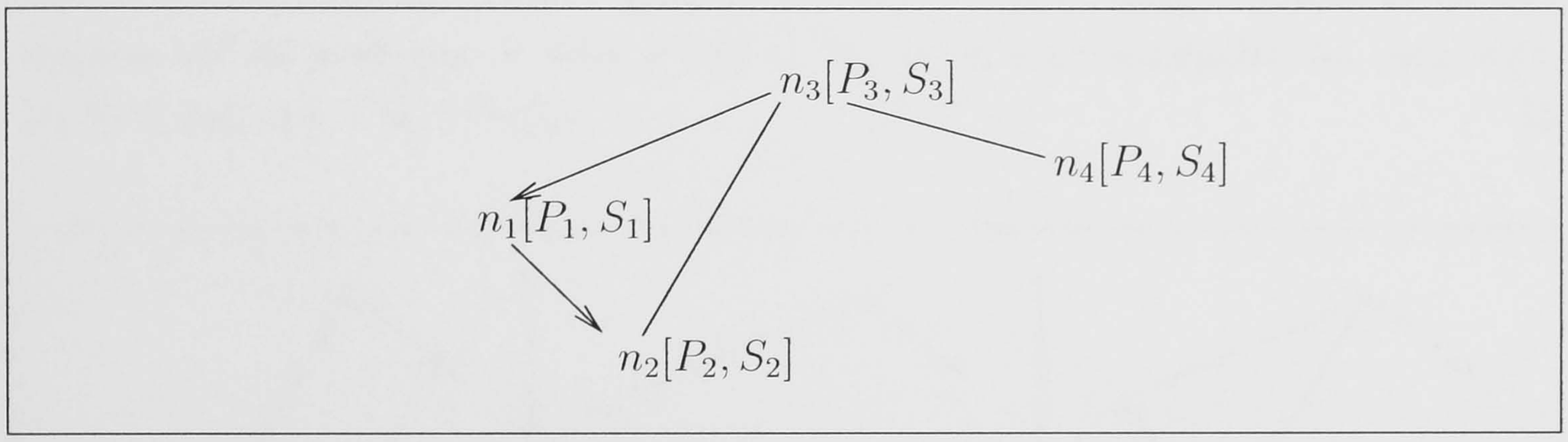

Figure 3.1: Example 3.5: connectivity graph

Not all pairings of connectivity graphs and networks are sensible. The following definition gives a minimal condition.

Definition 3.6 (Admissibility) A connectivity graph G is said to be *admissible* on a network N if $V(N) \subseteq V(G)$. ⊠

In the definition, recall that the vertex function V is overloaded and can be applied to networks (as defined in §3.1) and graphs alike. We use inclusion rather then equality for the sets of vertices as this will allow us to compare networks without having to require identical sets of vertices but only non-empty intersections of these. Throughout the text, if a connectivity graph is occurring in the context of a particular network but not explicitly defined, we assume that it is admissible on this network.

3.2.2 Network Topologies

In the envisioned scenario of mobile ad-hoc networks, network connectivity is subject to constant change. While connectivity graphs provide a means to represent snapshots of the network connectivity, the following definition allows for the description of connectivity properties which remain invariant over time.

Definition 3.7 (Network Topology) A *network topology* $\mathcal{T}$ is a non-empty collection of connectivity graphs. $\mathcal{T}$ is said to be *admissible* on a network N if all $G \in \mathcal{T}$ are. The *maximal network topology* $\mathcal{T}_{max}$ contains all graphs on $\mathcal{N}_{loc}$. ⊠

Example 3.8 For $i \in \{1,2,3\}$, let G_i be a connectivity graph with $V(G_i) = V_0 = \bigcup_{j=1}^{4} n_j$ and the following set of edges:

$$\begin{aligned} E(G_1) &= \{\{n_1,n_2\},\{n_2,n_3\},\{n_3,n_4\}\} \\ E(G_2) &= \{\{n_1,n_3\},\{n_3,n_4\}\} \\ E(G_3) &= \{\{n_1,n_2\},\{n_1,n_3\},\{n_2,n_3\},\{n_3,n_4\}\} \end{aligned}$$

Figure 3.2 depicts the network topology $\mathcal{T}_1 = \{G_1, G_2, G_3\}$. We can make the following observations: n_3 and n_4 are always connected; n_2 and n_4 are never connected; n_2 and n_1 are connected iff n_2 and n_3 are connected. Let furthermore $G_4 = (\{n_1,n_2,n_5\},\{\{n_1,n_2\},\{n_2,n_5\}\})$ and $\mathcal{T}_2 = \mathcal{T}_1 \cup \{G_4\}$. While $\mathcal{T}_1$ is admissible on all networks N with $V(N) \subseteq V_0$, $\mathcal{T}_2$ is only admissible on such with $V(N) \subseteq \{n_1,n_2\} = V_0 \cap V(G_4)$. ⊠

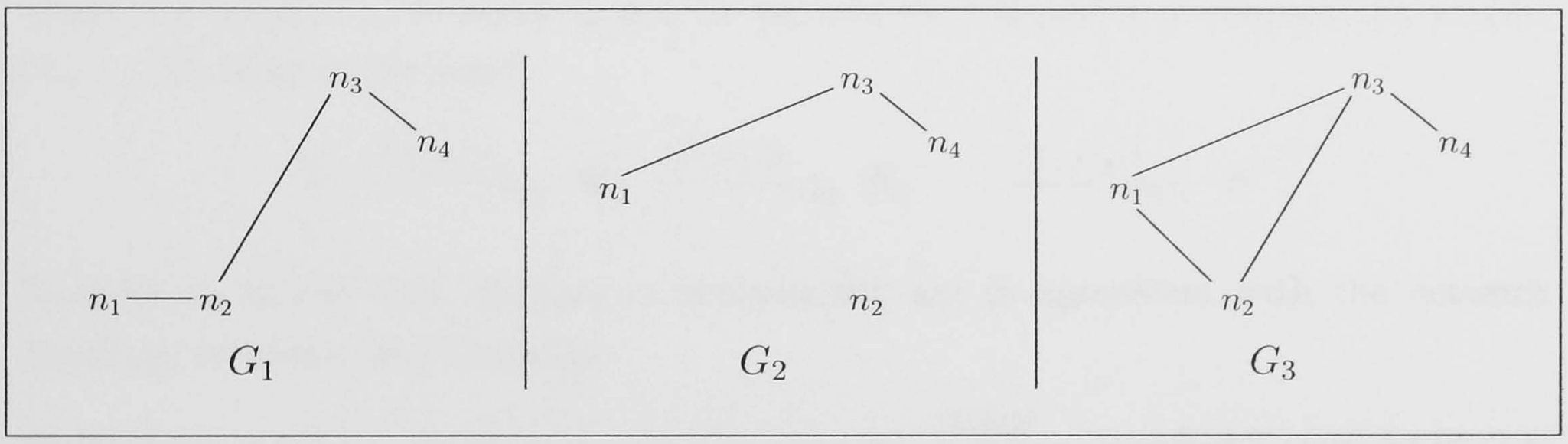

Figure 3.2: Example 3.8: network topology

3.2.3 Transition Relations

The operational semantics comprises three transition relations which we first give an overview of and then define formally.

Overview. The operational semantics is defined via the following transition relations:

$N \xrightarrow{(U,m)\sharp}_G N'$	labelled transition relation
$N \rightarrow N'$	reduction relation
$N \equiv N'$	structural equivalence

The *labelled transition relation* $N \xrightarrow{(U,m)\sharp}_G N'$ describes the evolution of a network N to a network N' during sending of the ground term U by node m, where N *abides by* a connectivity graph G: only nodes n with $(m,n) \in E(G)$ may receive U. The explanation of the labelling requires the definition of a synchronisation algebra [134]:

Definition 3.9 (Mode Identifiers) The set $\{!, ?, :\}$ is called the set of *mode identifiers*. They can be composed with the operator $\circ$ according to the algebra shown below, where $\perp$ means that ! cannot be combined with itself.

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

Mode identifiers are ranged over by the meta variables $\sharp$ and $\natural$. ⊠

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

The relation $\xrightarrow{(U,m)!}_G$ describes network evolution during the sending of exactly one term U, where the current connectivity is represented by G. The sending of k terms is serialised in the sense that k derivations for a sequence of connectivity graphs $G_1, \ldots, G_k$ have to be found:

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

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

Definition 3.10 ($\mathcal{T}$-Faithfulness) The relation $\xrightarrow{(U,m)\sharp}_G$ is called *$\mathcal{T}$-faithful* iff $G \in \mathcal{T}$, and we write $\xrightarrow{(U,m)\sharp}_{G\in\mathcal{T}}$ to emphasise this fact. ⊠

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

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

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

While the labelled transition relation defines the meaning of actions for external communication and makes them observable through the use of structured labels, the *reduction relation* $\rightarrow$ is concerned with the meaning of internal actions such as storage and retrieval.

The reduction relation is a prerequisite for the definition of *structural equivalence* $\equiv$, because if N reduces to N', we regard N and N' as structurally equivalent. Otherwise, structural equivalence is used to rearrange networks in order to prepare them for transitions of the labelled transition relation. Structural equivalence is trivially a congruence, i.e. $N \equiv N'$ implies $C[N] \equiv C[N']$ for every context C, because the only contexts for networks are of the form $[\,.\,] \parallel M$ and $M \parallel [\,.\,]$.

After this overview of the transition relations, we turn now to the presentation of their defining rules.

NIL	$n[\mathbf{nil}, S] \xrightarrow{(U,m):}_G n[\mathbf{nil}, S]$
OUT$_1$	$n[\mathbf{out}\ T.\,P, S] \xrightarrow{(T,n)!}_G n[P, S]$
OUT$_2$	$n[\mathbf{out}\ T.\,P, S] \xrightarrow{(U,m):}_G n[\mathbf{out}\ T.\,P, S]$
IN$_1$	$\dfrac{(m,n) \in E(G)}{n[\mathbf{in}\ x.\,P, S] \xrightarrow{(U,m)?}_G n[P[U/x], S]}$
IN$_2$	$\dfrac{(m,n) \notin E(G)}{n[\mathbf{in}\ x.\,P, S] \xrightarrow{(U,m):}_G n[\mathbf{in}\ x.\,P, S]}$
REPL$_1$	$n[!P, S] \xrightarrow{(U,m):}_G n[!P, S]$
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 the composition $\circ$ is due to Definition 3.9
RED	$\dfrac{N \rightarrow M \quad M \xrightarrow{(U,m)\sharp}_G M' \quad M' \rightarrow N'}{N \xrightarrow{(U,m)\sharp}_G N'}$
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 3.4: Labelled transition relation

Labelled transition relation. The rules for the labelled transition relation are defined in Table 3.4.

NIL A node n running the nil process $n[\mathbf{nil}, S]$ evolves to itself and loses any term U sent by any node m.

OUT$_1$ and OUT$_2$ These rules describe the behaviour of a sender $n[\mathbf{out}\ T.\,P, S]$. Such a node evolves to $n[P, S]$ when sending term T, expressed by labelling the transition arrow with $(T, n)!$. On the other hand, the sender evolves to itself in case of incoming terms U from any sender m, which are then lost.

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

REPL$_1$ A node running the replication process $n[!P, S]$ evolves to itself and loses any term U sent by any node m. Note that REPL$_2$ (reduction relation) provides for the expected expansion $!P = P \mid !P$. REPL$_1$ is necessary because the labelled transition relation examines every process in a parallel composition (see rules PPAR and PAR) and would get stuck on $!P$ as this process never disappears.

PPAR The rule for parallelism PPAR makes use of the algebra for the composition $\circ$ of $\sharp_1, \sharp_2 \in \{!, ?, :\}$ which is due to Definition 3.9. This is essential for the working of the broadcast modelling, as it makes sure that every parallel process running at every node in the network decides about receiving or losing a particular sent term. It is easy to see that the properties "$! \circ ! = \bot$" (i.e., $! \circ !$ is not allowed) and "$\sharp_1 \circ \sharp_2 \in \{?, :\} \Rightarrow \sharp_1, \sharp_2 \neq !$" of the algebra in Definition 3.9 ensure that at most one process can execute a sending action in any transition step.

RED and STRUCT These rules allow interleaved reduction and restructuring steps before and after a labelled transition relation.

The following example illustrates the modelling of local broadcast and the application of rules PPAR, OUT$_1$, IN$_1$, and IN$_2$.

Example 3.11 Let three nodes be defined as follows

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

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

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

If on the other hand $\mathcal{T} = \{G_1, G_2\}$, where $E(G_2) = \{(n_1, n_3)\}$, then we can conclude that the final configuration will either be $n_1[\textbf{nil}, S_1] \parallel n_2[\textbf{in}\ x.\,\textbf{nil}, S_2] \parallel n_3[\textbf{nil}, S_3]$ or $n_1[\textbf{nil}, S_1] \parallel n_2[\textbf{nil}, S_2] \parallel n_3[\textbf{in}\ x.\,\textbf{nil}, S_3]$. Note that $\mathcal{T}$ models the particular scenario in which either n_2 or n_3 are able to receive T. If we were to model full mobility instead (so that also both or neither of the nodes n_2 and n_3 might receive T), additional network connectivity graphs would have to be added to $\mathcal{T}$. ⊠

Reduction relation. The rules for the reduction relation are defined in Table 3.5.

REPL$_2$ $n[!P, S]$ reduces to $n[P \mid !P, S]$, thereby producing one copy of process P. The interplay with RED and STRUCT makes sure that as many copies of P as needed can be produced.

REPL$_2$	$n[!P, S] \;\rightarrow\; n[P \mid !P, S]$
STORE	$n[\textbf{store}\ T.\, P, S] \;\rightarrow\; n[P, S \cup T]$
READ	$\dfrac{T \in S}{n[\textbf{read}\ x.\, P, S] \;\rightarrow\; n[P[T/x], S]}$
CASE$_1$	$\dfrac{T = f(\widetilde{T}, \widetilde{U}) \qquad \lvert\widetilde{U}\rvert = \lvert\widetilde{x}\rvert}{n[\textbf{case}\ T\ \textbf{of}\ f(\widetilde{T}; \widetilde{x})\ P_1\ \textbf{else}\ P_2, S] \;\rightarrow\; n[P_1[\widetilde{U}/\widetilde{x}], S]}$
CASE$_2$	$\dfrac{\nexists \widetilde{U}.\, T = f(\widetilde{T}, \widetilde{U}) \wedge \lvert\widetilde{U}\rvert = \lvert\widetilde{x}\rvert}{n[\textbf{case}\ T\ \textbf{of}\ f(\widetilde{T}; \widetilde{x})\ P_1\ \textbf{else}\ P_2, S] \;\rightarrow\; n[P_2, S]}$
REFL-RED	$N \;\rightarrow\; N$
COMP-RED	$\dfrac{N_1 \;\rightarrow\; N_1'}{N_1 \parallel N_2 \;\rightarrow\; N_1' \parallel N_2}$

Table 3.5: Reduction relation

STORE By adding a term T to its store S, $n[\textbf{store}\ T.\, P, S]$ reduces to $n[P, S \cup T]$.

READ $n[\textbf{read}\ x.\, P, S]$ reduces to $n[P[T/x], S]$ for an arbitrary term $T \in S$.

CASE$_1$ and CASE$_2$ In the rules for case distinction, the term T is matched against a template $f(\widetilde{T}; \widetilde{x})$. If T is of the form $f(\widetilde{T}, \widetilde{U})$ and $\lvert\widetilde{U}\rvert = \lvert\widetilde{x}\rvert$, the continuation substitutes $\widetilde{U}$ for $\widetilde{x}$ in P_1 to yield $n[P_1[\widetilde{U}/\widetilde{x}], S]$. Otherwise, P_2 is the continuation process.

REFL-RED and COMP-RED These rules establish reflexivity and compositionality of the reduction relation.

Example 3.12 Assume the following definition:

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

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

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

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

Structural equivalence. The rules for the structural equivalence are defined in Table 3.6.

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-STRUCT	$N \equiv N$
SYM	$\dfrac{N' \equiv N}{N \equiv N'}$
TRANS	$\dfrac{N \equiv N'' \quad N'' \equiv N'}{N \equiv N'}$
COMP-STRUCT	$\dfrac{N_1 \equiv N_1'}{N_1 \parallel N_2 \equiv N_1' \parallel N_2}$
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 3.6: Structural equivalence

COMM through TRANS These rules provide standard definitions of commutativity and associativity for parallel compositions of networks, and establish reflexivity, symmetry, and transitivity of the structural equivalence.

COMP-STRUCT This rule says that structural changes in only one of the branches of parallel composition imply structural equivalence of the composed system.

PAR Rule PAR says that two parallel processes running at location n can be viewed as two separate nodes at n which run these processes and have access to the contents of the original store as necessary. Note that this rule is central to modelling internal communication, because one process might add a term to the store and PAR can make sure that this term is available to a process which runs in parallel (see also Example 3.13).

Example 3.13 Suppose

$$N \stackrel{\text{def}}{=} n[\textsf{store}\ T.\,\textsf{nil} \mid \textsf{read}\ x.\,\textsf{!out}\ x.\,\textsf{nil}, \varepsilon].$$

The following chain of equivalences holds:

$$\begin{array}{rcll}
N & \equiv & n[\textsf{store}\ T.\,\textsf{nil}, \varepsilon] \parallel n[\textsf{read}\ x.\,\textsf{!out}\ x.\,\textsf{nil}, \varepsilon] & \text{(by PAR)} \\
& \to & n[\textsf{nil}, \varepsilon \cup T] \parallel n[\textsf{read}\ x.\,\textsf{!out}\ x.\,\textsf{nil}, \varepsilon] & \text{(by COMP-RED, STORE)} \\
& \equiv & n[\textsf{nil} \mid \textsf{read}\ x.\,\textsf{!out}\ x.\,\textsf{nil}, \varepsilon \cup T] & \text{(by PAR)} \\
& \equiv & n[\textsf{nil}, \varepsilon] \parallel n[\textsf{read}\ x.\,\textsf{!out}\ x.\,\textsf{nil}, T] & \text{(by PAR)} \\
& \to & n[\textsf{!out}\ T.\,\textsf{nil}, T] \parallel n[\textsf{nil}, \varepsilon] & \text{(by COMM, COMP-RED, READ)} \\
& \to & n[\textsf{out}\ T.\,\textsf{nil} \mid \textsf{!out}\ T.\,\textsf{nil}, T] \parallel n[\textsf{nil}, \varepsilon] & \text{(by COMP-RED, REPL}_2\text{)} \\
& \equiv & N' \stackrel{\text{def}}{=} n[\textsf{out}\ T.\,\textsf{nil}, \varepsilon] \parallel n[\textsf{!out}\ T.\,\textsf{nil}, T] \parallel n[\textsf{nil}, \varepsilon] & \text{(by COMP-STRUCT, PAR)}
\end{array}$$

The derivation shows that the interplay of structural equivalence and reduction can be used by parallel processes for internal communication via the store, in this case of the value T. Continuing this example, the established structural equivalence $N \equiv N'$ can be used to derive a labelled transition for N, via the rules STRUCT and RED (note that we use the compressed notation $N \rightarrow / \equiv N'$ to express this):

$$\dfrac{N \rightarrow / \equiv N' \quad \dfrac{n[\mathbf{out}\ T.\mathbf{nil}, \varepsilon] \xrightarrow{(T,n)!}_G n[\mathbf{nil}, \varepsilon] \quad \cdots \xrightarrow{(T,n):}_G \cdots}{N' \xrightarrow{(T,n)!}_G N'' \stackrel{\text{def}}{=} n[\mathbf{nil}, \varepsilon] \parallel n[!\mathbf{out}\ T.\mathbf{nil}, T] \parallel n[\mathbf{nil}, \varepsilon]} \quad N'' \rightarrow / \equiv N''}{N \xrightarrow{(T,n)!}_G N''}$$

⊠

3.3 Mediated Equivalence

In this section, we define a notion of equivalence for networks in order to identify or to discriminate processes according to their behaviour. As in many other calculi, the operational semantics of CBS$^\sharp$ distinguishes between external and internal behaviour: the labelled transition relation defines observable interactions, while the reduction relation describes hidden computations. In the π-calculus for example, a similar distinction gives rise to two kinds of equivalences: one that abstracts from internal actions (called a *weak* equivalence), and one that does not (called a *strong* equivalence). For CBS$^\sharp$, it turns out that also the converse, i.e. abstracting from external actions and focusing on internal actions, can give a valuable notion of equivalence. This is because in wireless networks attackers can easily generate external actions by sending terms themselves, but only triggering internal actions in other nodes can be regarded a security breach. We thus adopt the notion of observation predicates from π-calculus in order to make *internal* actions visible, and use it to introduce *mediated equivalence* which on the one hand abstracts from *external* actions, on the other hand can fine-tune the visibility of internal actions.

Observation predicates have been introduced in the realm of π-calculus [96] (see §2.1.1) in order to add capabilities to observe some properties of the state a process is in. This becomes necessary if the transition system does not provide sufficient observables itself, e.g. in the case of CHAM style reduction semantics. In the π-calculus, observation predicates which determine the channels on which communication actions can occur are found to give the right notion. We apply these ideas to CBS$^\sharp$, with the goal of observing the states of store components which turn out to be a crucial ingredient to security analysis in Chapter 5.

In Table 3.7, we thus define a *barb predicate* $N \downarrow_n U$ which will help to discriminate networks by their capability to store terms. Intuitively, $N \downarrow_n U$ holds if N can "immediately" store term U at location n, i.e. without requiring another network interaction. The meaning of the rules can be described as follows:

BARB-EMPTY	$N \downarrow_n \varepsilon$
BARB-STORE	$n[\textbf{store}\ T.\,P, S] \downarrow_n T$
BARB-STRUCT	$\dfrac{N \equiv N' \quad N' \downarrow_n U}{N \downarrow_n U}$
BARB-PPAR	$\dfrac{N \downarrow_n U}{N \parallel M \downarrow_n U}$

Table 3.7: Barb predicate

BARB-EMPTY This rule states that any network N can store the empty term ε.

BARB-STORE The only other axiom of the predicate says that $n[\textbf{store}\ T.\,P, S]$ has the capability to store T at n.

BARB-STRUCT Other steps of structural equivalence are allowed before U is stored.

BARB-PPAR The store action may also happen in only one part of the network.

In order to define the desired equivalence, the barb predicate has to be complemented by two additional concepts:

Convergence The focus is on storage actions which will *eventually* happen during the evolution of a network; the influence of the observability of communication actions is lessened.

Mediation The relevance of storage actions can be fine-tuned according to the type of term stored.

Convergence has been introduced in conjunction with the development of testing equivalence for the Spi Calculus [4] (see §2.1.4) to determine whether a process will eventually exhibit a certain channel. Our adaption of this notion is motivated by a desired security property (see Chapter 5) which regards only 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). The *convergence predicate* $N \Downarrow_n^{\mathcal{T}} U$ is defined in Table 3.8 and holds intuitively if N will eventually store U at location n, possibly after some interactions under network topology $\mathcal{T}$. The meaning of the rules of the convergence predicate can be described as follows:

CONV-BARB Network N converges to store U if it can immediately store U.

$$\textsc{Conv-Barb} \quad \frac{N \downarrow_n U}{N \Downarrow_n^{\mathcal{T}} U}$$

$$\textsc{Conv-Comm} \quad \frac{N \xrightarrow{(U,m)!}_{G \in \mathcal{T}} N' \quad N' \Downarrow_n^{\mathcal{T}} U}{N \Downarrow_n^{\mathcal{T}} U}$$

Table 3.8: Convergence predicate

Conv-Comm This rule allows for an arbitrary number of communication actions of N before converging to store U. Note the mode identifier ! which expresses that all the communications have to be initiated by N itself.

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

Definition 3.14 A *mediator* μ is a function on terms $\mu : \mathbf{T} \to \mathbf{T}$. A mediator μ is called *simple* if $\mu(U) \in \{\varepsilon, U\}$ for all U. ⊠

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

Definition 3.15 (Mediated Equivalence) For given topology $\mathcal{T}$ and mediator μ, we write $N \sqsubseteq_{\mathcal{T}}^{\mu} M$ whenever

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

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

$$N \simeq_{\mathcal{T}}^{\mu} M \;\; \textit{iff} \;\; N \sqsubseteq_{\mathcal{T}}^{\mu} M \text{ and } M \sqsubseteq_{\mathcal{T}}^{\mu} N. \qquad ⊠$$

Sometimes it is helpful to consider families of mediators in the following way:

Definition 3.16 (Mediation by Families of Mediators) Let $(\mu_n)_{n \in \mathcal{N}}$ be a family of mediators. We write $N \sqsubseteq_{\mathcal{T}}^{(\mu_n)_{n \in \mathcal{N}}} M$ whenever

$$N \Downarrow_n^{\mathcal{T}} U \text{ implies } M \Downarrow_n^{\mathcal{T}} \mu_n(U) \text{ for each barb } U \text{ and } n \in V(N) \cap V(M). \qquad ⊠$$

The next example illustrates how the mediator can be used to fine-tune the equivalence to a certain specification.

Example 3.17 Let a mediator μ_S and two networks N and $M(U)$ be defined as follows.

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

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

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

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

This example also indicates that mediated equivalence might be used for security reasoning: to see this, let the terms stored by N represent routing table entries which describe valid (in the case of T_{sec}) or invalid (T_{attack}) statements about the network topology; to be considered secure, N has to run a protocol which can determine the validity of a term, and this ability can be checked by letting μ_S express when a term is considered valid. This kind of security reasoning is investigated in much more depth in §5.2, §5.3, and Chapter 6, where further examples are provided.

We conclude this section with the proof of the following helpful theorem about the relation of networks N and $N \parallel M$ which has been suggested by Example 3.17:

Theorem 3.18 *For networks N, M and simple mediator μ, the following holds:*

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

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

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

Fix thus $U \neq \varepsilon$ and $n \in V(N)$, assume $N \Downarrow_n^{\mathcal{T}} U$. Because μ is simple (see Definition 3.14), we can do a case distinction on the value of $\mu(U)$. If $\mu(U) = \varepsilon$, we can show $N \parallel M \Downarrow_n^{\mathcal{T}} \mu(U)$ to hold directly with Conv-Barb and Barb-Empty.

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

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

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

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

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

3.4 Notation

This section concludes the chapter with the presentation of some notational conventions which are helpful to compile large specifications in §3.4.1, and examples of cryptographic modelling in §3.4.2.

3.4.1 Syntactic Translations

For the specification of large protocols as undertaken in Chapter 6, clarity and readability of the formalisation are crucial. This is because of the well-known weakness of model-based verification efforts: while the verification system helps to establish a property about a model with certainty, the modelling itself can oftentimes only be validated by inspection. For this reason, we have opted for keywords such as **in** and **read** rather than just symbols which are favoured by other calculi, and we have introduced a convention to use intelligible expressions for various syntactic constructs (see Table 3.2). In this section, we introduce more conventions which will prove helpful in the presentation of the examples.

Notation 3.19 We introduce the following syntactic translations:

(1) **if** $T = U$ **then** P_1 **else** P_2 $\equiv$ **case** Match(T) **of** Match$(U;)$ P_1 **else** P_2
(2) **in** $f(\widetilde{T};\widetilde{x}).P$ $\equiv$ **in** $x.$ **case** x **of** $f(\widetilde{T};\widetilde{x})$ P **else nil**, where $x \notin fv(P)$
(3) **read** $f(\widetilde{T};\widetilde{x})$ P_1 **else** P_2 $\equiv$ **read** $x.$ **case** x **of** $f(\widetilde{T};\widetilde{x})$ P_1 **else** P_2, where $x \notin fv(P_1)$

⊠

Notation (1) shows that the action for case distinction is expressive enough to encode if-then-else where the condition tests equality of terms $T = U$. This is done with the help of the special function Match since case can only match terms which a function has been applied to. Recall that all terms before the semicolon in a function matching have to be equal to their counterparts in the term to be matched. Thus Match$(U;)$ forces equality of T and U.

Notations (2) and (3) define a simple translation for receive and retrieve actions which are directly followed by a case distinction on their input variable. This happens frequently as one often wants to distinguish between sent or stored terms of different formats and handle them differently. The translations are straightforward, note however that the newly introduced variable should be fresh with respect to the free variables of the continuations to avoid variable capture.

Notation 3.20 Trailing **nil**s can be omitted whenever no confusion arises. ⊠

Notation 3.21 Line breaks and indentation may be used in the specification of processes. ⊠

Notation 3.22 We have already used definitions such as $P \stackrel{\text{def}}{=} \textbf{out}\ n.\,\textbf{nil}$ or $N \stackrel{\text{def}}{=} n[P, S]$ to abbreviate long process specifications. For convenience, we may also use definitions parametrised with names, such as $P(n) \stackrel{\text{def}}{=} \textbf{out}\ n.\,\textbf{nil}$ and $N(n) \stackrel{\text{def}}{=} n[P(n), S]$, with the intuitive meaning, e.g. $P(m) = \textbf{out}\ m.\,\textbf{nil}$ and $N(m) = m[\textbf{out}\ m.\,\textbf{nil}, S]$. Definitions must not occur recursively or mutually recursively. ⊠

Notation 3.23 In our implementation (see §4.5), we use the keywords **process** and **network** to syntactically distinguish between definitions of different types, e.g. $\textbf{process}\ P(n) \stackrel{\text{def}}{=} \textbf{out}\ n.\,\textbf{nil}$. ⊠

3.4.2 Cryptographic Modelling

The terms of CBS$^\sharp$ can be flexibly structured with the help of functions, as shown in §3.1. While this turns out to be helpful for various modelling tasks, for the purpose of security analysis cryptographic modelling is most important. In the following we present modelling suggestions for the usual cryptographic primitives that will be used throughout this dissertation.

Symmetric Encryption. The encryption of a term U under symmetric key n (shared-key cryptography) can be modelled as $\mathsf{SymEnc}(n, U)$. We have to make sure that only processes which possess key n can decrypt the new term. This can be achieved by the following two rules during specification:

(1) distributing n only to the processes which are supposed to share the key; and,

(2) not allowing case distinction on terms $\mathsf{SymEnc}(n, U)$ except in the following form:

$$\begin{aligned}&\textbf{symdec}\ T\ \mathit{symkey}\ x\ P_1\ \textbf{else}\ P_2 \equiv \\ &\quad \textbf{case}\ T\ \textbf{of}\ \mathsf{SymEnc}(\mathit{symkey}\colon x)\ P_1\ \textbf{else}\ P_2\end{aligned}$$

The properties of case distinction make sure that the encrypted content is only available in P_1 if *symkey* evaluates to the correct symmetric key which must be in the possession of the respective process.

Asymmetric Encryption. The style of specification used to express symmetric encryption can be applied analogously to asymmetric encryption (public-key cryptography). Key pairs are specified as

$$(\mathsf{PubKey}(seed), \mathsf{PrivKey}(seed))$$

created from the same *seed* by applying functions PubKey and PrivKey. An encryption of a term U under public key PubKey(*seed*) then simply corresponds to applying the function AsymEnc to yield AsymEnc(PubKey(*seed*), U). Decrypting amounts to verifying that the seeds of the private key and the public key used for the encryption are the same. As shown in the following definition, this procedure can be completely hidden in the protocol specification by definition of the action **asymdec**, where *seed* is a fresh variable, $seed \notin fv(P_1)$.

asymdec T *privkey* x P_1 **else** $P_2 \equiv$
  case *privkey* **of** PrivKey(; *seed*)
    case T **of** AsymEnc(PubKey(*seed*); x) P_1 **else** P_2
  else
    P_2

Rules (1) and (2) stated above for symmetric encryption apply analogously in this situation.

Digital Signatures. Public-key digital signatures can use essentially the same mechanisms we have introduced for asymmetric encryption. A signature of the terms $\widetilde{T}$ under private key PrivKey(*seed*) corresponds to applying the function Sign to yield Sign(PrivKey(*seed*), $\widetilde{T}$). Checking of the signature can be done as follows, where *seed* is again a fresh variable, $seed \notin fv(P_1)$:

checksig T *pubkey* $[\widetilde{T}]$ P_1 **else** $P_2 \equiv$
  case *pubkey* **of** PubKey(; *seed*)
    case T **of** Sign(PrivKey(*seed*), $\widetilde{T}$;) P_1 **else** P_2
  else
    P_2

For clarity, the elements $\widetilde{T}$ are enclosed in square brackets.

Public Key Infrastructure. A public key infrastructure (PKI) is an arrangement which provides for the certification of bindings of public keys to users, usually through communication with a trusted third party. Sometimes it is convenient to be able to assume the existence of a PKI without modelling all the details. This can be modelled elegantly with the help of the introduced mechanisms for public-key cryptography by using its name n as seed in the public/private key generation. For example, if node n signs a

term it will do so with PrivKey(n). Likewise a node m which is to check a signature supposedly done by n just applies function PubKey to n and uses this key as input to **checksig**.

Chapter 4

Control Flow Analysis of CBS$^\sharp$

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. In this chapter we apply this technique to CBS$^\sharp$ in order to determine the sets of values generated by communication and storage in networks. More precisely, the result of our analysis for a network N yields an overapproximation of

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

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

The benefit of having such an analysis becomes apparent by the properties the analysis exhibits with respect to the behavioural semantics of CBS$^\sharp$: under certain conditions, the analysis allows for checking whether networks are mediated equivalent. This means that proofs of mediated equivalence can be derived automatically (i.e., if not prevented by overapproximation) using the implementation of the analysis which is also presented and proved correct in this chapter.

This chapter is in five sections. In §4.1 we give an overview of the fundamentals of control flow analysis and we present flow logic as a specification framework for such analyses. We specify our analysis as a flow logic in §4.2 where we also introduce a static abstraction of the network topology $\mathcal{T}$ as an important step to limit the state space arising from network execution. Classical correctness results for the analysis are shown in §4.3. In §4.4 other important properties of the analysis are presented which characterise the analysis estimates and the relation between the analysis and behavioural semantics of CBS$^\sharp$. In §4.5, we show that the analysis can be implemented using an off-the-shelf constraint solving tool.

4.1 Background: Control Flow Analysis

Control flow and data flow analyses [104] are known as techniques to extract semantic information of programs at compile time. *Control flow analysis* has been developed

to determine the execution paths of a program, originally in the setting of functional languages [128]. Oftentimes, this control flow information is a necessary precursor for *data flow analysis*, which is used in optimising compilers. In recent years, the control flow and data flow analysis technology has also been applied more frequently to the analysis of concurrent systems represented as terms of some process calculus, e.g. [23, 105, 20, 109]. In this setting, neither the term control flow nor data flow is applicable in its original sense, and analyses usually perform both aspects. However, it has become customary to uniformly use the term control flow analysis to denote these kinds of analyses.

Oftentimes, a control flow analysis can be expressed as a *constraint-based analysis* [104, 6], a program analysis technique which separates between two tasks: *constraint generation*, which specifies the desired information about the program, and *constraint resolution*, which computes the desired information. This separation is one of the main advantages of this technique, as it leads to clarity and therefore better understanding of the analysis.

In the following we present flow logic as a specification style for the constraint generation part of constraint-based analyses. We employ this style in §4.2.

Specification. The term *flow logic* has been coined by [103, 108] to denote a notational style for specifying analyses, applicable to various programming paradigms. By abstracting from specific constraint specification formalisms and using standard mathematical notations, flow logic has the advantage of a meta-language which can be used to present an analysis without requiring additional knowledge about particular formalisms. If needed, the thus specified analysis can be "compiled down" to a proper constraint specification language of sufficient expressivity. However, the proof of correctness of this conversion process can be complicated and involves potentially several intermediate flow logics [26].

For a program or process $P \in \mathbf{P}$ in some language, a flow logic expresses the acceptability of an analysis estimate $\Theta \in \mathbf{A}$ via a judgement $\Theta \models P$, which is defined by rules of the following form:

$$\Theta \models P \ \textit{iff} \ \text{"}\textit{some conditions involving } \Theta, P, \models \textit{ hold}\text{"}$$

In general, these rules have to be interpreted coinductively, so that an analysis estimate can only be rejected if it does not fulfil the conditions of the rule. This is because the right-hand side logic formula may mention $\Theta' \models P'$ for P' which are not subterms of P; the flow logic is then called *abstract*. In many cases however, the flow logic will be *compositional* (i.e., syntax-directed), so that P' will always be a subterm of P, and in this case the coinductive and inductive interpretations coincide (this is the case for the flow logics presented in this dissertation). Orthogonally to the classification into

abstract and compositional specifications, flow logics are also categorised into succinct or verbose style. An analysis which has one global estimate Θ, such as shown above, is called *verbose*. It is called *succinct* in the case there is an additional local component θ, which refers only to the expression P in question, and is written $\Theta \models P : \theta$. Occurrences of components of θ on the right-hand side of rules are implicitly existentially quantified. As shown in [108], succinct specifications can always be reduced to verbose ones, but are often preferred due to their conciseness.

Correctness. The correctness of a flow logic is determined by proving the following properties [104, 108]:

(1) Well-definedness

(2) Semantic correctness

(3) Moore-family result

In order to establish well-definedness, note that the rules define an analysis functional $\mathcal{A} : \mathbf{Q} \to \mathbf{Q}$ for a complete lattice $(\mathbf{Q}, \sqsubseteq)$, namely the set $\mathbf{A} \times \mathbf{P} \to \{tt, ff\}$ with the ordering:

$$Q_1 \sqsubseteq Q_2 \;\; \textit{iff} \;\; \forall\, (\Theta, P) \in \mathbf{A} \times \mathbf{P}.\; Q_1(\Theta, P) = tt \Rightarrow Q_2(\Theta, P) = tt$$

Following the coinductive interpretation of the rules, the analysis is called *well-defined* if the functional $\mathcal{A}$ has a greatest fixed point. This can be guaranteed by showing that $\mathcal{A}$ is monotonic, so that it follows from Tarski's fixed point theorem that it has a complete lattice of fixed points in $\mathbf{Q}$.

Intuitively, an analysis is *semantically correct* if the analysis estimate contains information about the entire evolution of P, as described by its formal semantics R. This is usually expressed by a subject reduction theorem:

$$\text{if } \Theta \models P \text{ and } (P, P') \in R \text{ then } \Theta \models P'$$

So if Θ is an acceptable analysis estimate for P, and P evolves to P' according to the formal semantics described by relation R, then Θ is an acceptable analysis estimate for P' as well.

It is also reasonable to ask whether each program P (a) admits an acceptable analysis estimate and (b) has a best analysis estimate. Both questions can be answered by proving the *Moore-family result*. A subset X of a complete lattice L is called a Moore family iff $\bigsqcap Y \in X$ for all $Y \subseteq X$. The Moore-family result is then stated as follows:

$$\{\Theta \;:\; \Theta \models P\} \text{ is a Moore family for all } P \in \mathbf{P}$$

By definition, a Moore family is never empty, which shows (a), and $\bigsqcap\{\Theta : \Theta \models P\}$ is a least solution, showing (b).

Implementation. We have introduced flow logic as a meta-language for specifying constraint generation in constraint-based analysis. Once translated into a particular constraint specification formalism, the constraint resolution part can be performed efficiently using dedicated *constraint solvers*. For a particular flow logic, a constraint solver should be chosen according to the type of constraints it expects as input, as this determines whether the analysis is expressible in the language of the solver, and whether the translation can be easily proved correct. To provide a small overview, we mention the following tools that have been developed in recent years: the *Berkeley Analysis Engine (BANE)* [7] and its successor *Banshee* [83] are efficient solvers for set constraints; the solver *Reqs* [77] is a generic solver for recursive equations system; the *Succinct Solver* [107, 102] computes the least interpretation of a given formula in a Horn-like fragment of first-order predicate logic. We have chosen the Succinct Solver as constraint solving tool for our implementation and thus describe it in more detail in §4.5.1.

4.2 Specification

In this section, we use the flow logic paradigm in order to specify a control flow analysis for CBS$^\sharp$ which overapproximates the sets of terms that may be sent and stored in a given network. As a prerequisite for a safe approximation, §4.2.1 presents an abstraction for network topologies. In §4.2.2 we then define the judgements of the analysis.

4.2.1 Network Topology Abstractions

According to Definition 3.10, the $\mathcal{T}$-faithful 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 $\mathcal{T}$ as follows:

Definition 4.1 (Abstraction Function for Network Topologies) The function $\mathcal{G}$, defined as

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

is called the *abstraction function* for network topologies. ⊠

This means that an *abstract network topology* $\widehat{G} = \mathcal{G}(\mathcal{T})$ is again a connectivity graph, and contains all $G \in \mathcal{T}$ as subgraphs.

Notation 4.2 In our implementation (see §4.5), we use the notation

$$\textbf{topology}\ [(n, n') \mid \{n, n'\}]^{+}$$

in order to specify an abstract connectivity graph with unidirectional and/or bidirectional links. ⊠

It also ensures that an analysis over the abstract network topology will enable rule IN_1 whenever a $G \in \mathcal{T}$ would have done it, since the following holds:

Fact 4.3 $(m, n) \in E(\mathcal{G}(\mathcal{T}))$ *iff* $\exists\, G \in \mathcal{T}.\ (m, n) \in E(G)$

4.2.2 Judgements

We define a syntax-directed analysis with two judgements to range over the syntactic categories of processes and networks:

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

The *judgement for networks* $(\kappa, \sigma) \models_{\widehat{G}} N$ reads "(κ, σ) is a valid analysis estimate describing the behaviour of N under abstract network topology $\widehat{G}$". It is parametrised with $\widehat{G}$ and yields the following sets of values, where $\mathbf{T}$ is defined as in Table 3.1, but could be restricted to contain ground terms only:

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

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 $(T, n) \in \sigma$.

Notation 4.4 We may sometimes write $T \in \kappa_n$ to denote $(T, n) \in \kappa$, and analogously $T \in \sigma_n$. ⊠

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

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

We use the substitution with the notations introduced in §3.1, in particular $\theta[U/x]$ to denote the function which equals θ except at x which is mapped to U. Furthermore, the empty substitution environment is denoted []. The substitution θ is introduced as a parameter to the judgement for processes mainly for clarity and to facilitate the proof of the correctness of the implementation (see §4.5.3). It expresses that the process P is to be analysed under the substitution θ, and indeed the analysis could be equivalently rewritten in the following way, which can be proved by a straightforward structural induction on P:

$$(\kappa, \sigma) \models^{\theta}_{\widehat{G},n} P \ \ \textit{iff} \ \ (\kappa, \sigma) \models^{[]}_{\widehat{G},n} P\theta$$

After this overview of the judgements, we turn to the formal specification of the analysis as a flow logic, the rules of which we explain in the following.

Judgement for Processes. The rules for the judgement for processes $(\kappa, \sigma) \models^{\theta}_{\widehat{G},n} P$ are defined in Table 4.1.

CFA-NIL The analysis of the terminated process holds trivially for any analysis components.

CFA-OUT (κ, σ) is a valid analysis of process **out** $T.P$ under substitution θ iff the following holds: the network cache κ contains $(T\theta, n)$, as $T\theta$ is precisely the term which can be sent by this process at location n; furthermore, (κ, σ) has to be a valid analysis for the continuation process P under substitution θ.

CFA-IN For the analysis of the reception **in** $x.P$, all (U, m) recorded in κ have to be examined, that is all terms U sent from any location m in the network. If there is an edge $(m, n) \in E(\widehat{G})$, i.e. the term U might have been received at n, continuation P is analysed under the new substitution $\theta[U/x]$ which takes into account that U might have been substituted for x.

CFA-STORE This rule is similar to CFA-OUT. However, instead of inserting $(T\theta, n)$ into the network cache, the store cache is updated, $(T\theta, n) \in \sigma$, since the analysis corresponds to the storage action **store** $T.P$.

CFA-READ This rule resembles CFA-IN, with the difference that pairs (U, n) are now taken from the store cache σ instead of the network cache, since we deal with retrieval of terms from the store with the action **read** $x.P$. The continuation P is again evaluated under the updated substitution.

Judgement for Processes			
CFA-Nil	$(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n}$ **nil**		
CFA-Out	$(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n}$ **out** $T.P$	*iff*	$(T\theta, n) \in \kappa \wedge$ $(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n} P$
CFA-In	$(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n}$ **in** $x.P$	*iff*	$\forall\, (U,m) \in \kappa.\ (m,n) \in E(\widehat{G}) \Rightarrow$ $(\kappa,\sigma) \vDash^{\theta[U/x]}_{\widehat{G},n} P$
CFA-Store	$(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n}$ **store** $T.P$	*iff*	$(T\theta, n) \in \sigma \wedge$ $(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n} P$
CFA-Read	$(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n}$ **read** $x.P$	*iff*	$\forall\, (U,n) \in \sigma.\ (\kappa,\sigma) \vDash^{\theta[U/x]}_{\widehat{G},n} P$
CFA-Case	$(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n}$ **case** T **of** $f(T_1,\ldots,T_j;x_{j+1},\ldots,x_k)\ P_1$ **else** P_2	*iff*	$(T\theta = f(U_1,\ldots,U_k) \wedge \bigwedge_{i=1}^{j} T_i\theta = U_i \Rightarrow$ $(\kappa,\sigma) \vDash^{\theta[U_i/x_i]_{i=j+1}^{k}}_{\widehat{G},n} P_1) \wedge$ $(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n} P_2$
CFA-Par	$(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n} P_1 \mid P_2$	*iff*	$(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n} P_1 \wedge (\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n} P_2$
CFA-Repl	$(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n}$ **!**P	*iff*	$(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n} P$
Judgement for Networks			
CFA-Node	$(\kappa,\sigma) \vDash_{\widehat{G}} n[P\theta, S]$	*iff*	$(\kappa,\sigma) \vDash^{\theta}_{\widehat{G},n} P \wedge$ $\forall\, U \in S.\ (U,n) \in \sigma$
CFA-PPar	$(\kappa,\sigma) \vDash_{\widehat{G}} N_1 \parallel N_2$	*iff*	$(\kappa,\sigma) \vDash_{\widehat{G}} N_1 \wedge (\kappa,\sigma) \vDash_{\widehat{G}} N_2$

Table 4.1: Control Flow Analysis for CBS$^\sharp$

CFA-Case In the analysis of **case** T **of** $f(T_1,\ldots,T_j;x_{j+1},\ldots,x_k)\ P_1$ **else** P_2, it is checked whether $T\theta$, the term to be matched, is of the form $f(U_1,\ldots,U_k)$. Furthermore, it is ensured that $T_1\theta = U_1 \wedge \ldots \wedge T_j\theta = U_j$, meaning that the first j arguments of f match. Then the continuation P_1 is analysed under a substitution which additionally maps x_i to U_i for $i = j+1,\ldots,k$, meaning the remaining arguments of f are bound to the variables x_i. For safety, both branches of the case statement have to be analysed, and therefore P_2 is evaluated under the substitution θ.

CFA-Par An analysis of the parallel composition $P_1 \mid P_2$ under substitution θ is valid iff the analyses of both branches are valid under θ, respectively.

CFA-REPL For the analysis of the replication $!P$ under substitution θ it suffices to analyse the process P once under substitution θ.

Judgement for Networks. The rules for the judgement for networks $(\kappa, \sigma) \models_{\widehat{G}} N$ are defined in Table 4.1.

CFA-NODE (κ, σ) is a valid analysis estimate describing the behaviour of node $n[P\theta, S]$ under abstract connectivity graph $\widehat{G}$ iff it is also a valid analysis result for process P at n under substitution θ, and for all terms U contained in store S it holds that $(U, n) \in \sigma$.

CFA-PPAR An analysis of the parallel composition $N_1 \parallel N_2$ requires analyses of both the branches.

In order to illustrate the working of the analysis, we continue Example 3.12 where nodes make use of the data structure $\mathsf{Env}(n; text)$, to send content $text$ in an envelope with address n:

Example 4.5 Consider the following network N under network topology $\mathcal{T} = \{G_1, G_2\}$, where $E(G_1) = \{(n_1, n_2)\}$ and $E(G_2) = \{(n_1, n_3)\}$:

$$\begin{array}{lcl} N_1 & \stackrel{\text{def}}{=} & n_1[\textbf{in } x.\, \textbf{case } x \textbf{ of } \mathsf{Env}(n_1; y) \textbf{ store } y.\, \textbf{nil else nil}, \varepsilon] \\ N_2 & \stackrel{\text{def}}{=} & n_2[\textbf{out } \mathsf{Env}(n_1, text_2).\, \textbf{nil}, \varepsilon] \\ N_3 & \stackrel{\text{def}}{=} & n_3[\textbf{out } \mathsf{Env}(n_2, text_3).\, \textbf{nil}, \varepsilon] \\ N & \stackrel{\text{def}}{=} & N_1 \parallel N_2 \parallel N_3 \end{array}$$

We are interested in computing the constraints for valid analysis estimates (κ, σ) for N with the given network topology. The abstraction of $\mathcal{T}$ yields $E(\mathcal{G}(\mathcal{T})) = \{(n_1, n_2), (n_1, n_3)\}$. Because of CFA-PPAR, we have

$$(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N \;\; \textit{iff} \;\; \bigwedge_{i=1}^{3} (\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N_i.$$

We first compute the constraints for N_2:

$$\begin{array}{lll} & (\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N_2 & \\ \textit{iff} & (\kappa, \sigma) \models^{[]}_{\mathcal{G}(\mathcal{T}), n_2} \textbf{out } \mathsf{Env}(n_1: text_2).\, \textbf{nil} \wedge (\varepsilon, n_2) \in \sigma & \text{(by CFA-NODE)} \\ \textit{iff} & (\mathsf{Env}(n_1, text_2), n_2) \in \kappa \wedge (\kappa, \sigma) \models^{[]}_{\mathcal{G}(\mathcal{T}), n_2} \textbf{nil} \wedge (\varepsilon, n_2) \in \sigma & \text{(by CFA-OUT)} \\ \textit{iff} & (\mathsf{Env}(n_1, text_2), n_2) \in \kappa \wedge (\varepsilon, n_2) \in \sigma & \text{(by CFA-NIL)} \end{array}$$

Analogously for N_3: $(\mathsf{Env}(n_2, text_3), n_3) \in \kappa \wedge (\varepsilon, n_3) \in \sigma$. In the case of N_1, we know by CFA-NODE that $(\varepsilon, n_1) \in \sigma$ and the remaining constraints imposed by CFA-NODE are computed as follows:

$(\kappa, \sigma) \models^{[]}_{\mathcal{G}(\mathcal{T}), n_1}$ **in** x. **case** T **of** $\mathsf{Env}(n; x)$ **store** x. **nil else nil**

iff $\forall\, (U, m) \in \kappa.\ (m, n_1) \in E(\mathcal{G}(\mathcal{T})) \Rightarrow$

$(\kappa, \sigma) \models^{[U/x]}_{\mathcal{G}(\mathcal{T}), n_1}$ **case** x **of** $\mathsf{Env}(n_1; y)$ **store** y. **nil else nil** (by CFA-IN)

iff $\forall\, (U, m) \in \kappa.\ (m, n_1) \in E(\mathcal{G}(\mathcal{T})) \Rightarrow$

$U = \mathsf{Env}(U_1, U_2) \wedge n_1 = U_2 \Rightarrow (\kappa, \sigma) \models^{[U/x][U_2/y]}_{\mathcal{G}(\mathcal{T}), n_1}$ **store** y. **nil** (by CFA-CASE)

iff $\forall\, (U, m) \in \kappa.\ (m, n_1) \in E(\mathcal{G}(\mathcal{T})) \Rightarrow$

$U = \mathsf{Env}(U_1, U_2) \wedge n_1 = U_2 \Rightarrow (U_2, n_1) \in \sigma$ (by CFA-STORE, CFA-NIL)

It can be easily verified that a least solution to the complete constraint system for (κ, σ) is given by:

$$(\mathsf{Env}(n_1, text_2), n_2) \in \kappa \wedge (\varepsilon, n_2) \in \sigma \wedge (\mathsf{Env}(n_2, text_3), n_3) \in \kappa \wedge (\varepsilon, n_3) \in \sigma \wedge$$
$$(\varepsilon, n_1) \in \sigma \wedge (text_2, n_1) \in \sigma$$

⊠

4.3 Correctness

In §4.1 we have introduced a set of generic properties which determine the correctness of a flow logic. In this section we show that the analysis of Table 4.1 enjoys these properties.

Well-Definedness. According to §4.1, a flow logic is well-defined if it has a greatest fixed point.

Theorem 4.6 (Well-Definedness) *The judgement* $(\kappa, \sigma) \models_{\widehat{G}} N$ *is well-defined.*

Proof. Since the judgement is defined compositionally, a simple structural induction shows that the analysis functional it defines is monotonic. Thus the functional possesses a lattice of fixed points by Tarski's fixed point theorem. □

Furthermore, the compositional specification entails that the functional has exactly one fixed point so that the least and the greatest fixed points coincide.

Semantic Correctness. Semantic correctness with respect to the operational semantics of §3.2 is stated as a subject reduction theorem (Theorem 4.9). Informally, it expresses that the analysis estimate remains acceptable when the network evolves.

Before we can turn to this main result, we need the following lemma stating auxiliary subject reduction results which hold for the reduction relation and structural equivalence.

Lemma 4.7

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

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

Proof. Both parts are shown by induction on the inference of $N \rightarrow N'$ and $N \equiv N'$, respectively. The full proofs can be found in Appendix A.1 on page 147. □

Another lemma is needed in order to simplify the proofs of Theorem 4.9 (and later also Theorem 4.11). The lemma states that if network N can send term T from location n, then the message cache κ computed by an analysis of N contains (T, n).

Lemma 4.8 *If* $N \xrightarrow{(T,n)!}_{G \in \mathcal{T}} N'$ *and* $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$, *then* $(T, n) \in \kappa$.

Proof. The result can be shown by induction on the inference of $N \xrightarrow{(T,n)!}_{G \in \mathcal{T}} N'$. The full proof can be found in Appendix A.1 on page 149. □

Using Lemma 4.7 and 4.8, we can prove the main semantic correctness theorem, which ensures that the analysis estimate is a safe description of what will happen during the evolution of a network.

Theorem 4.9 (Subject Reduction)

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

Proof. The result can be shown by induction on the inference of $M \xrightarrow{(U,m)\sharp}_{G \in \mathcal{T}} M'$ for subterms M and M' of N and N', respectively. The full proof can be found in Appendix A.1 on page 150. □

Moore-Family Result. Recall the definition of a Moore family from §4.1: a subset X of a complete lattice L is called a Moore family iff $\bigsqcap Y \in X$ for all $Y \subseteq X$. Proving that the set of all acceptable analysis estimates is a Moore family for all networks N enables us to state that there is a "least" analysis estimate for all N. Here, "least" is to be understood with respect to the following partial order:

$$(\kappa_1, \sigma_1) \sqsubseteq (\kappa_2, \sigma_2) \ \textit{iff} \ \kappa_1 \subseteq \kappa_2 \wedge \sigma_1 \subseteq \sigma_2$$

Using this ordering, we can prove the following theorem:

Theorem 4.10 $\{(\kappa, \sigma) : (\kappa, \sigma) \models_{\widehat{G}} N\}$ *is a Moore family for all* $N \in \mathbf{N}$.

Proof. To prove the theorem, assume that for some index set I and $N \in \mathbf{N}$ the following holds:

$$\forall\, i \in I.\ (\kappa_i, \sigma_i) \models_{\widehat{G}} N$$

It then remains to show that $\prod_{i\in I}(\kappa_i, \sigma_i) \models_{\widehat{G}} N$. This can be done by a structural induction on N and the help of the following auxiliary result (proved by structural induction on P):

$$\{(\kappa, \sigma) \ : \ (\kappa, \sigma) \models^{\theta}_{\widehat{G},n} P\} \textit{ is a Moore family for all } P \in \mathbf{P}$$

Both inductions are straightforward, we will thus only show two exemplary cases of the proof of the auxiliary result.

Case out $T.P$. Assume $\forall\ i \in I.\ (\kappa_i, \sigma_i) \models^{\theta}_{\widehat{G},n}$ **out** $T.P$. Thus by rule CFA-OUT we have $(T\theta, n) \in \kappa_i$ and $(\kappa_i, \sigma_i) \models^{\theta}_{\widehat{G},n} P$ for all $i \in I$. We can use the induction hypothesis to yield $\prod_{i\in I}(\kappa_i, \sigma_i) \models^{\theta}_{\widehat{G},n} P$. Since $(T\theta, n) \in \kappa_i$ for all $i \in I$, we have $(T\theta, n) \in \bigcap_{i\in I} \kappa_i$. Therefore we can use the reverse direction of CFA-OUT to establish the desired $\prod_{i\in I}(\kappa_i, \sigma_i) \models^{\theta}_{\widehat{G},n}$ **out** $T.P$.

Case in $x.P$. In order to show $\prod_{i\in I}(\kappa_i, \sigma_i) \models^{\theta}_{\widehat{G},n}$ **in** $x.P$, it suffices by rule CFA-IN to show:

$$\forall\ (U, m) \in \bigcap_{i\in I} \kappa_i.\ (m, n) \in E(\widehat{G}) \Rightarrow \prod_{i\in I}(\kappa_i, \sigma_i) \models^{\theta[U/x]}_{\widehat{G},n} P$$

Thus fix $(U, m) \in \bigcap_{i\in I} \kappa_i$ $(*)$ and assume $(m, n) \in E(\widehat{G})$ $(**)$. Because of $(*)$, we have $(U, m) \in \kappa_i$ $(***)$ for all $i \in I$. We have by assumption $(\kappa_i, \sigma_i) \models^{\theta}_{\widehat{G},n}$ **in** $x.P$ for all $i \in I$, thus rule CFA-IN in conjunction with $(***)$ and $(**)$ gives $(\kappa_i, \sigma_i) \models^{\theta[U/x]}_{\widehat{G},n} P$ for all $i \in I$. We can then apply the induction hypothesis to have $\prod_{i\in I}(\kappa_i, \sigma_i) \models^{\theta[U/x]}_{\widehat{G},n} P$ which concludes the proof. □

4.4 Properties

In §4.3 we have established general correctness properties for our analysis such as subject reduction and existence of least solutions. In order to show the usefulness of the analysis, more specific properties are needed. Therefore, in §4.4.1, we formally establish what elements a valid analysis estimate contains. In §4.4.2, we go one step further and prove that, under certain conditions, the analysis results can be used to automatically establish mediated equivalence of certain networks.

4.4.1 Characterisation of the Analysis Estimate

In §4.2 we have informally stated what elements a valid estimate (κ, σ) of our Control Flow Analysis will contain. The goal of this section is to formally establish these statements.

The two main results describing the contents of a valid analysis estimation almost directly follow from subject reduction (Theorem 4.9). 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 4.11 *If* $N \longrightarrow_{\mathcal{T}}^{*} N' \xrightarrow{(T,n)!}_{G \in \mathcal{T}} N''$ *and* $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$, *then* $(T, n) \in \kappa$.

Proof. We have $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$ by assumption and can thus apply Theorem 4.9 finitely many times to the derivations $N \longrightarrow_{\mathcal{T}}^{*} N'$ to yield $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N'$. With $N' \xrightarrow{(T,n)!}_{G \in \mathcal{T}} N''$ we can apply Lemma 4.8 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 4.12 *For* $T \neq \varepsilon$, *the following implication holds: if* $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$ *and* $N \Downarrow_n^{\mathcal{T}} T$, *then* $(T, n) \in \sigma$.

Proof. By induction on the inference of $N \Downarrow_n^{\mathcal{T}} T$, which involves only finitely many applications of CONV-COMM, CONV-BARB, BARB-PPAR, BARB-STRUCT, and BARB-STORE, there exist N', N'', P, S such that the following statements hold:

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

Since we have $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$ by assumption, we can thus apply Theorem 4.9 finitely many times to the derivations $N \longrightarrow_{\mathcal{T}}^{*} N'$ to yield $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N'$. We can apply Lemma 4.7 (2) to have $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N'' \parallel n[\textbf{store}\ T.P, S]$. Furthermore, by CFA-PPAR we have $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} n[\textbf{store}\ T.P, S]$, and we can apply CFA-STORE as in the corresponding case of the proof of Lemma 4.7 (1) to have $(T, n) \in \sigma$. □

A direct logical consequence of Theorem 4.12 is the following corollary:

Corollary 4.13 *For* $T \neq \varepsilon$, *the following implication holds: if* $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N$ *and* $(T, n) \notin \sigma$, *then* $N \not\Downarrow_n^{\mathcal{T}} T$.

4.4.2 Relation to the Behavioural Semantics of CBS$^\sharp$

Proving the mediated equivalences by hand tends to be error-prone for larger protocols. The following theorem shows that we can use the analysis to establish mediated equivalence under certain conditions:

Theorem 4.14 *For all networks* N, M *and simple mediators* μ, *the following implication holds: if* $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N \parallel M$ *and* $\forall\, (U, m) \in \sigma.\ \mu(U) = \varepsilon$, *then* $N \simeq_{\mathcal{T}}^{\mu} N \parallel M$.

Proof. By Definition 3.15, we know $N \simeq_{\mathcal{T}}^{\mu} N \parallel M$ iff $N \sqsubseteq_{\mathcal{T}}^{\mu} N \parallel M$ and $N \parallel M \sqsubseteq_{\mathcal{T}}^{\mu} N$. Since μ is simple, i.e. $\mu(U) \in \{\varepsilon, U\}$ for all U, the first inclusion can be proved by Theorem 3.18.

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

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

To prove this by contradiction, we assume its negation:

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

Since $\neg(N \Downarrow^{\mathcal{T}}_{n} \mu(U))$ it must be that $\mu(U) \neq \varepsilon$ $(*)$, because $N \Downarrow^{\mathcal{T}}_{n} \varepsilon$ is true for any N. $(*)$ allows us to apply Theorem 4.12 to $N \parallel M \Downarrow^{\mathcal{T}}_{n} U$ and $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N \parallel M$, both of which we have by assumption, to yield $(U, n) \in \sigma$. However, we know by assumption that $\forall\, (U', m) \in \sigma.\, \mu(U') = \varepsilon$, in particular $\mu(U) = \varepsilon$. This is a contradiction to $(*)$. $\Box$

An analogous result can be established for families of simple mediators as well:

Corollary 4.15 *For all networks N, M and a family of simple mediators $(\mu_n)_{n \in \mathcal{N}}$, the following implication holds: if $(\kappa, \sigma) \models_{\mathcal{G}(\mathcal{T})} N \parallel M$ and $\forall\, (U, m) \in \sigma.\, m \in V(N) \Rightarrow \mu_m(U) = \varepsilon$, then $N \simeq^{(\mu_n)_{n \in \mathcal{N}}}_{\mathcal{T}} N \parallel M$.*

4.5 Implementation

We have seen in §4.1 that flow logic provides a framework to abstractly specify control flow analyses and prove properties about them. Both specification and proofs are facilitated by the fact that flow logics are very close to the formal semantics of the language to be analysed but, as a trade-off, the effective computation of the analysis might not be straightforward. Indeed, the analysis specified in §4.2 is uncomputable in general, simply because it is specified over an infinite set of terms. It is however possible to define a finite analysis that is safe with respect to the original analysis. This development is undertaken in this section.

In §4.5.1 we describe the Succinct Solver, the constraint solver we are using for the computation of our analysis, and we give an introduction to its constraint language ALFP. In §4.5.2 we specify a finite version of our analysis, not yet as a constraint generation algorithm but again as a flow logic. Defining such an intermediary flow logic has the advantage that, because of the similarity of the formalisms, the finite analysis can be proved correct with relatively moderate effort (in §4.5.3); on the other hand the actual constraint generation algorithm of §4.5.4 can be derived directly from it. Finally, in §4.5.5, we investigate precision and complexity of the implemented analysis.

4.5.1 Background: The Succinct Solver

The Succinct Solver [107, 102] is a constraint solving tool based on a solver algorithm for the constraint language *Alternation-free Least Fixpoint Logic (ALFP)*. ALFP is a fragment of first order predicate logic which is more expressive [107] than for example Datalog [32], but allows for polynomial model checking routines. The clauses cl of ALFP are built from variables x, constants c, function symbols f, predicates R, terms t, and preconditions pre according to the following grammar [102]:

$$\begin{array}{lcl} t & ::= & c \mid x \mid f(t_1, \ldots, t_k) \\ pre & ::= & R(t_1, \ldots, t_k) \mid \neg R(t_1, \ldots, t_k) \mid t_1 = t_2 \mid t_1 \neq t_2 \\ & & \mid pre_1 \wedge pre_2 \mid pre_1 \vee pre_2 \mid \forall\, x.\ pre \mid \exists\, x.\ pre \\ cl & ::= & R(t_1, \ldots, t_k) \mid \mathbf{1} \mid cl_1 \wedge cl_2 \mid \forall\, x.\ cl \mid pre \Rightarrow cl \end{array}$$

The clauses are interpreted over a universe $\mathcal{U}$ of ground terms. Satisfaction relations

$$(\rho, \sigma) \models pre \text{ and } (\rho, \sigma) \models cl$$

are defined in the standard way, where ρ is an interpretation of predicate symbols and σ an interpretation of terms. For example, a clause $R(x_1, \ldots, x_k)$ is interpreted according to the following rule:

$$(\rho, \sigma) \models R(x_1, \ldots, x_k) \ \ \mathit{iff}\ \ (\sigma(x_1), \ldots, \sigma(x_k)) \in \rho(R)$$

Given a clause with no free variables the Succinct Solver computes the least interpretation ρ of predicate symbols that makes the clause true. The least interpretation always exists since for an interpretation σ_0 of free variables in a clause cl, the set of all ρ with $(\rho, \sigma_0) \models cl$ is a Moore family [107]. Solvability is however subject to a notion of stratification (all clauses presented in this dissertation are trivially stratified because we do not use negated relations in preconditions).

The Succinct Solver actually comprises two solving engines: in V1.0, the logic is interpreted over a finite universe without function symbols, in V2.0 the universe can be expanded dynamically at the expense of admitting a slightly smaller logic (no universal quantification in preconditions). Note that our analysis does not make use of the advanced features of V2.0 since termination of the solving process is only ensured if the expansion of the universe of ground terms terminates. This is not necessarily the case for our analyses, since for example an attacker process might try to produce infinitely long terms. We therefore encode function symbols as predicates and terms by finite abstractions as explained in detail in §4.5.2.

4.5.2 Finite Analysis of Labelled CBS$^\sharp$

The analysis of §4.2 is specified over an infinite set of terms **T**. However, in order for the analysis to be guaranteed to terminate, interpretation over a finite universe is required. The standard solution to this problem is to find a finite representation for infinite sets of values and manipulate these representations during the analysis accordingly. Grammars [80, 6] and tree automata [87] have been used as candidates for such finite representations. Tree grammars in particular (see e.g. [37]) have been employed before in the context of control flow analysis, for instance in [106]. In this section we develop a finite analysis based on this concept which can be proved correct with respect to the original analysis (see §4.5.3) and can directly be converted into a constraint generation algorithm for ALFP clauses (see §4.5.4). Before describing the judgements of this analysis, we give a summary of tree grammars and discuss our labelling of CBS$^\sharp$.

Background: Tree Grammars. Recall the definition of the set of terms **T** for CBS$^\sharp$ in Table 3.1. A term $T \in \mathbf{T}$ can be viewed as an ordered tree where the leaves are variables $x \in \mathcal{X}$, names $n \in \mathcal{N}$, or function symbols $f \in \mathcal{F}$ with $arity(f) = 0$, and where the internal nodes are function symbols with a positive arity equalling their out-degree. Thus, **T** and similar term languages can be described by *tree grammars* [80, 37] which can be compared to classical word grammars [72] except that they have trees as their basic objects.

Formally, a tree grammar Γ is a tuple (N, Σ, R, S) of a set of nonterminal symbols N, a set of terminal symbols Σ with given arities, a set of production rules R, and a start symbol $S \in N$. Tree grammars are then classified by the structure of the rules. We are concerned only with *normalised regular* tree grammars for which all non-terminal symbols have arity zero and rules are of the form $A \rightarrow a(A_1, \ldots, A_k)$ (denoted $a(A_1, \ldots, A_k) \in R(A)$), where $A \in N$, $A_i \in N$ for $i = 1, \ldots, k$, and $a \in \Sigma$ with $arity(a) = k$. The language generated by a normalised regular tree grammar is defined as follows:

Definition 4.16 Let $\Gamma = (N, \Sigma, R, S)$ be a normalised regular tree grammar. The language generated by Γ is denoted $L(R, S)$, where N and Σ are assumed fixed, and defined as follows:

$$L(R, S) = \{a(t_1, \ldots, t_k) \ : \ a(A_1, \ldots, A_k) \in R(S) \wedge \bigwedge_{i=1}^{k} t_i \in L(R, A_i)\}$$

⊠

Labelling. In order to obtain a finite analysis, we introduce a labelling of the parts of the syntax of CBS$^\sharp$ which are used in the analysis estimate (κ, σ). This means attaching labels to all terms occurring in a given network and refining the analysis to work over

tree grammars with labels as nonterminal symbols. Intuitively, instead of recording potentially infinitely many elements (U, n) for actions **out** $T.P$ and **store** $T.P$, the refined analysis records (ℓ, n), where ℓ is the label of T, and grammar rules which produce a set containing all U for starting symbol ℓ.

Elements of κ or σ are pairs of a ground term and a location. It therefore suffices to introduce a labelling on ground terms; variables do not need to be labelled. Thus, let $\mathcal{L}$ be a countably infinite set of *labels* and let ℓ range over $\mathcal{L}$. The set of labelled terms is defined as follows:

$$T^\ell ::= n^\ell \mid x \mid f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k})$$

Labelled CBS$^\sharp$ is obtained from CBS$^\sharp$ by replacing ordinary terms with labelled terms, and the function *labelling* is defined to yield a labelling for networks N, processes P, and terms T of CBS$^\sharp$. The function *labelling* can be chosen arbitrarily, but the choice influences the precision of the analysis: a unique labelling is most precise, and assigning the same label to every term is maximally imprecise. In between these extremes, a heuristics suggests itself which we are using to compute our examples (see page 92).

The function *labels* yields the set of labels of a term T^ℓ, process P, or network N of Labelled CBS$^\sharp$. The function $\lfloor . \rfloor : \mathbf{T}^\ell \to \mathbf{T}$ is used to invert this function.

Judgements. The finite analysis makes use of the following components to replace κ, σ, and θ:

$\kappa^\star$	$\subseteq$	$\mathcal{L} \times \mathcal{N}_{loc}$	finite message cache
$\sigma^\star$	$\subseteq$	$\mathcal{L} \times \mathcal{N}_{loc}$	finite store cache
γ			tree grammar production rules
$\theta^\star$	:	$\mathcal{X} \to \mathcal{L}$	finite substitution

In contrast to their counterparts, the finite components collect labels instead of terms. γ is a set of production rules that is part of a tree grammar $(\mathcal{L}, \mathcal{F} \cup \mathcal{N}, \gamma, \ell)$ with $\ell \in \mathcal{L}$ and formally defining $arity(n) = 0$ for all $n \in \mathcal{N}$. Note that in the analysis of a network N, the judgements defined below make only use of labels $labels(N) \subseteq \mathcal{L}$, and since $labels(N)$ is finite for any network of finite size, the analysis is guaranteed to terminate. Termination is not guaranteed if the analysis is run over a domain of ground terms, in fact it would not terminate on the examples in Chapter 6, since the attacker might produce an infinite set of ground terms.

We define the function $lab^{\theta^\star} : \mathbf{T}^\ell \to \mathcal{L}$ to retrieve labels from any labelled term and to perform substitution for variables:

$$\begin{aligned}
lab^{\theta^\star}(n^\ell) &= \ell \\
lab^{\theta^\star}(x) &= \theta^\star(x) \\
lab^{\theta^\star}(f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k})) &= \ell
\end{aligned}$$

Using this definition, Table 4.4 specifies the following judgements and predicate:

$\gamma \models^{\theta^\star} T^\ell$	judgement for terms
$(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n} P$	judgement for processes
$(\kappa^\star, \sigma^\star, \gamma) \models_{\widehat{G}} N$	judgement for networks
$\mathcal{E}^{\theta^\star}_{\gamma}(\ell', T^\ell)$	equality predicate

$(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n} P$ and $(\kappa^\star, \sigma^\star, \gamma) \models_{\widehat{G}} N$ are the finite counterparts to the judgements for processes and networks of the original analysis, $\gamma \models^{\theta^\star} T^\ell$ is an auxiliary statement for labelled terms in order to build the production rules γ. $\mathcal{E}^{\theta^\star}_{\gamma}(\ell', T^\ell)$ is an auxiliary predicate which expresses that the languages $L(\gamma, \ell')$ and $L(\gamma, lab^{\theta^\star}(T^\ell))$ have a common element, i.e. the term labelled ℓ' and T might be equal.

Judgement for Terms. The judgement for terms $\gamma \models^{\theta^\star} T^\ell$ ensures that the production rules γ are built correctly.

FIN-NAME γ is a valid set of production rules for n^ℓ if it contains the rule $\ell \to n$, written $n \in \gamma(\ell)$.

FIN-VAR Variables are not labelled therefore any γ is valid for a variable x.

FIN-FUN In order to represent a function $f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k})$ correctly in γ, all subterms $T_i^{\ell_i}$ for $i = 1, \ldots, k$ have to be correctly represented first. Then the rule $\ell \to f(lab^{\theta^\star}(T_1^{\ell_1}), \ldots, lab^{\theta^\star}(T_k^{\ell_k}))$ is added to γ. Recall that $lab^{\theta^\star}$ yields either the outermost label of a function or name, or yields the label to be substituted for x according to $\theta^\star$.

Example 4.17 Consider the following judgement where $\theta^\star = [x \mapsto \ell_x, y \mapsto \ell_y]$:

$$\gamma \models^{\theta^\star} f^{\ell_f}(n^{\ell_n}, x, g^{\ell_g}(m^{\ell_m}, y))$$

Because of FIN-FUN, the set of rules in γ is enlarged by $\ell_f \to f(\ell_n, \ell_x, \ell_g)$ and furthermore (with FIN-NAME and FIN-FUN) by $\ell_n \to n$, $\ell_g \to g(\ell_m, \ell_y)$, and $\ell_m \to m$.

During the course of an analysis, assume that the judgement may be invoked a second time with $[x \mapsto \ell'_x, y \mapsto \ell'_y]$ instead of $\theta^\star$. If we furthermore require that rules $\ell_x \to a$, $\ell_y \to b$, $\ell'_x \to a'$, $\ell'_y \to b'$ are the only other rules in γ, the language associated with ℓ_f computes to:

$$L(\gamma, \ell_f) = \{f(n, a, g(m, b)), f(n, a', g(m, b'))\}$$ ⊠

Judgement for Processes. The judgement for processes $(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n} P$ is almost straightforward considering the original analysis. We describe three exemplary rules:

Judgement for Terms

FIN-NAME	$\gamma \models^{\theta^\star} n^\ell$	*iff* $n \in \gamma(\ell)$
FIN-VAR	$\gamma \models^{\theta^\star} x$	
FIN-FUN	$\gamma \models^{\theta^\star} f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k})$	*iff* $\bigwedge_{i=1}^k \gamma \models^{\theta^\star} T_i^{\ell_i} \wedge$ $f(\mathsf{lab}^{\theta^\star}(T_1^{\ell_1}), \ldots, \mathsf{lab}^{\theta^\star}(T_k^{\ell_k})) \in \gamma(\ell)$

Judgement for Processes

FIN-NIL	$(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n}$ **nil**	
FIN-OUT	$(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n}$ **out** $T^\ell.P$	*iff* $\gamma \models^{\theta^\star} T^\ell \wedge (\mathsf{lab}^{\theta^\star}(T^\ell), n) \in \kappa^\star \wedge$ $(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n} P$
FIN-IN	$(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n}$ **in** $x.P$	*iff* $\forall\, (\ell', m) \in \kappa^\star.\ (m, n) \in E(\widehat{G}) \Rightarrow$ $(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star[x \mapsto \ell']}_{\widehat{G},n} P$
FIN-STORE	$(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n}$ **store** $T^\ell.P$	*iff* $\gamma \models^{\theta^\star} T^\ell \wedge (\mathsf{lab}^{\theta^\star}(T^\ell), n) \in \sigma^\star \wedge$ $(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n} P$
FIN-READ	$(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n}$ **read** $x.P$	*iff* $\forall\, (\ell', n) \in \sigma^\star.\ (\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star[x \mapsto \ell']}_{\widehat{G},n} P$
FIN-CASE	$(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n}$ **case** T^ℓ **of** $f(T_1^{\ell_1}, \ldots, T_j^{\ell_j}; x_{j+1}, \ldots, x_k)\ P_1$ **else** P_2	*iff* $\gamma \models^{\theta^\star} T^\ell \wedge \bigwedge_{i=1}^j \gamma \models^{\theta^\star} T_i^{\ell_i} \wedge$ $(\forall\, \ell'_{j+1}, \ldots, \ell'_k.\ (\exists\, \ell'_1, \ldots, \ell'_j.$ $f(\ell'_1, \ldots, \ell'_k) \in \gamma(\mathsf{lab}^{\theta^\star}(T^\ell)) \wedge$ $\bigwedge_{i=1}^j \mathcal{E}_\gamma^{\theta^\star}(\ell'_i, T_i^{\ell_i}))\ \Rightarrow$ $(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star[x_i \mapsto \ell'_i]_{i=j+1}^k}_{\widehat{G},n} P_1) \wedge$ $(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n} P_2$
FIN-PAR	$(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n} P_1 \mid P_2$	*iff* $(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n} P_1 \wedge (\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n} P_2$
FIN-REPL	$(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n}\ !P$	*iff* $(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n} P$

Judgement for Networks

FIN-NODE	$(\kappa^\star, \sigma^\star, \gamma) \models_{\widehat{G}} n[P, S]$	*iff* $(\kappa^\star, \sigma^\star, \gamma) \models^{[]}_{\widehat{G},n} P \wedge$ $(\forall\, U^\ell \in S.\ \gamma \models^{[]} U^\ell \wedge (\mathsf{lab}^{[]}(U^\ell), n) \in \sigma^\star)$
FIN-PPAR	$(\kappa^\star, \sigma^\star, \gamma) \models_{\widehat{G}} N_1 \parallel N_2$	*iff* $(\kappa^\star, \sigma^\star, \gamma) \models_{\widehat{G}} N_1 \wedge (\kappa^\star, \sigma^\star, \gamma) \models_{\widehat{G}} N_2$

Equality Predicate

EQ-NAME	$\mathcal{E}_\gamma^{\theta^\star}(\ell', n^\ell)$	*iff* $L(\gamma, \ell') \cap L(\gamma, \ell) \neq \emptyset$
EQ-VAR	$\mathcal{E}_\gamma^{\theta^\star}(\ell', x)$	*iff* $L(\gamma, \ell') \cap L(\gamma, \mathsf{lab}^{\theta^\star}(x)) \neq \emptyset$
EQ-FUN	$\mathcal{E}_\gamma^{\theta^\star}(\ell', f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k}))$	*iff* $\exists\, \ell'_1, \ldots, \ell'_k.\ f(\ell'_1, \ldots, \ell'_k) \in \gamma(\ell') \wedge$ $\bigwedge_{i=1}^k \mathcal{E}_\gamma^{\theta^\star}(\ell'_i, T_i^{\ell_i})$

Table 4.4: Finite Analysis for Labelled CBS$^\sharp$

FIN-OUT In order for $(\kappa^\star, \sigma^\star, \gamma)$ to be a valid analysis for **out** $T^\ell.P$, the γ has to be a valid representation for T^ℓ. The remainder of the rule is analogous to CFA-OUT.

except that only the label $lab^{\theta^\star}(T^\ell)$ is recorded in $\kappa^\star$ instead of the full term.

FIN-IN This rule is completely analogous to CFA-IN, considered the change from terms to labels.

FIN-CASE It has to be ensured that γ is a valid representation of all occurring terms $T^\ell, T_1^{\ell_1}, \ldots, T_j^{\ell_j}$. Then all rules of the form $f(\ell'_1, \ldots, \ell'_k) \in \gamma(lab^{\theta^\star}(T^\ell))$ are examined (which are associated with the term T^ℓ to be matched). It is checked whether the terms associated with the labels of the first j components of $f(\ell'_1, \ldots, \ell'_k)$ might be equal with respect to $T_1^{\ell_1}, \ldots, T_j^{\ell_j}$, expressed by the equality predicate $\mathcal{E}_\gamma^{\theta^\star}(\ell'_i, T_i^{\ell_i}))$ for $i = 1, \ldots, j$. If all the equality predicates hold, the continuation P_1 is analysed, where $\theta^\star$ is updated with new bindings to labels $\ell'_{j+1}, \ldots, \ell'_k$ for variables $x_{j+1}, \ldots, x_k$.

Judgement for Networks. The judgement for networks $(\kappa^\star, \sigma^\star, \gamma) \models_{\widehat{G}} N$ is straightforward as well. We describe one rule:

FIN-NODE This rule is analogous to CFA-NODE. Because contents of the store S are labelled as well, it is ensured that γ is a valid rule set for them, and their labels are recorded in $\sigma^\star$.

Equality Predicate. The equality predicate $\mathcal{E}_\gamma^{\theta^\star}(\ell', T^\ell)$ expresses intuitively that the term labelled ℓ' and the term $\lfloor T^\ell \rfloor$ could be equal, i.e. more formally: the languages $L(\gamma, \ell')$ and $L(\gamma, lab^{\theta^\star}(T^\ell))$ have common elements.

EQ-NAME and EQ-VAR Both rules test whether the intersection of $L(\gamma, \ell')$ and $L(\gamma, lab^{\theta^\star}(T^\ell))$ is non-empty, where T is either a name or a variable.

EQ-FUN Similar to FIN-CASE, all rules of the form $f(\ell'_1, \ldots, \ell'_k) \in \gamma(\ell')$ are examined and it is recursively checked for $i = 1, \ldots, k$ whether the term labelled ℓ'_i and the term $\lfloor T_i^{\ell_i} \rfloor$ could be equal. Note that $\gamma \models^{\theta^\star} T_i^{\ell_i}$ holds for all $i = 1, \ldots, k$ as the equality predicate is only invoked for FIN-CASE where this has been established already.

4.5.3 Correctness

In this section we establish the correctness of the finite analysis (Table 4.4) with respect to the original analysis (Table 4.1). This is expressed by the main result in Theorem 4.21 which says that an analysis estimation $(\kappa^\star, \sigma^\star, \gamma)$ from $(\kappa^\star, \sigma^\star, \gamma) \models_{\widehat{G}} \mathit{labelling}(N)$ yields an analysis estimation (κ, σ) such that $(\kappa, \sigma) \models_{\mathcal{G}(T)} N$, provided that the analysis components are related in the expected way. Three auxiliary lemmas are needed first.

Lemma 4.18 *Let $\theta^\star$ be given and define for all $x \in \mathcal{X}$*

$$\theta(x) \in \{U \ : \ \exists\, \ell \in \mathcal{L}.\ \theta^\star(x) = \ell \wedge U \in L(\gamma, \ell)\}.$$

Then the following implication holds:

$$\textit{If } \gamma \models^{\theta^\star} T^\ell \textit{ then } \lfloor T^\ell \rfloor \theta \in L(\gamma, \mathrm{lab}^{\theta^\star}(T^\ell))$$

Proof. The result is shown by structural induction on T^ℓ. The full proof can be found in Appendix A.2 on page 151. □

Lemma 4.19 *Let $\theta^\star$ be given and let θ be defined as in Lemma 4.18. Then the following implication holds:*

$$\textit{If } \gamma \models^{\theta^\star} T^\ell \textit{ and } \lfloor T^\ell \rfloor \theta \in L(\gamma, \ell') \textit{ then } \mathcal{E}_\gamma^{\theta^\star}(\ell', T^\ell)$$

Proof. The result is shown by structural induction on T^ℓ. The full proof can be found in Appendix A.2 on page 151. □

Lemma 4.20 *Let $\kappa^\star$, $\sigma^\star$, and $\theta^\star$ be given and define*

$$\begin{array}{lcl} \kappa & = & \{(U,n) \,:\, \exists\, \ell \in \mathcal{L}.\ (\ell,n) \in \kappa^\star \wedge U \in L(\gamma,\ell)\} \\ \sigma & = & \{(U,n) \,:\, \exists\, \ell \in \mathcal{L}.\ (\ell,n) \in \sigma^\star \wedge U \in L(\gamma,\ell)\} \\ \theta(x) & \in & \{U \,:\, \exists\, \ell \in \mathcal{L}.\ \theta^\star(x) = \ell \wedge U \in L(\gamma,\ell)\} \textit{ for all } x \in \mathcal{X}. \end{array}$$

Then the following implications hold:

(1) *If* $\gamma \models^{\theta^\star} T^\ell$ *and* $(\mathrm{lab}^{\theta^\star}(T^\ell), n) \in \kappa^\star$ *then* $(\lfloor T^\ell \rfloor \theta, n) \in \kappa$

(2) *If* $\gamma \models^{\theta^\star} T^\ell$ *and* $(\mathrm{lab}^{\theta^\star}(T^\ell), n) \in \sigma^\star$ *then* $(\lfloor T^\ell \rfloor \theta, n) \in \sigma$

Proof. The proof can be found in Appendix A.2 on page 152. □

Theorem 4.21 *Let $\kappa^\star$, $\sigma^\star$, and $\theta^\star$ be given and let κ, σ, and θ be defined as in Lemma 4.20. Then the following implications hold:*

(1) *If* $(\kappa^\star, \sigma^\star, \gamma) \models^{\theta^\star}_{\widehat{G},n} \mathrm{labelling}(P)$ *then* $(\kappa, \sigma) \models^{\theta}_{\mathcal{G}(T),n} P$

(2) *If* $(\kappa^\star, \sigma^\star, \gamma) \models_{\widehat{G}} \mathrm{labelling}(N)$ *then* $(\kappa, \sigma) \models_{\mathcal{G}(T)} N$

Proof. Both parts are shown by structural induction on labelled processes and networks, respectively. The full proof can be found in Appendix A.2 on page 152. □

4.5.4 Constraint Generation Algorithm

In §4.5.1 we have reviewed the Succinct Solver, the constraint solver we use for computing the finite analysis. While we have defined and proved correct the finite analysis already, it remains to be shown that it can be formalised in ALFP, the constraint language used as input to the solver. This is however a straightforward process where

$$
\begin{array}{lcl}
\texttt{fun genTerm}\ \theta^\star\ (n^\ell) & = & \gamma(\ell, n) \\
\quad \mid \texttt{genTerm}\ \theta^\star\ x & = & \mathbf{1} \\
\quad \mid \texttt{genTerm}\ \theta^\star\ (f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k})) & = & \bigwedge_{i=1}^{k} \texttt{genTerm}\ \theta^\star\ (T_i^{\ell_i}) \wedge \\
 & & \gamma(\ell, \ell) \wedge f(\ell, lab^{\theta^\star}(T_1^{\ell_1}), \ldots, lab^{\theta^\star}(T_k^{\ell_k})) \\
\\
\texttt{fun genProc}\ n\ \theta^\star\ (\textbf{nil}) & = & \mathbf{1} \\
\quad \mid \texttt{genProc}\ n\ \theta^\star\ (\textbf{out}\ T^\ell.\, P) & = & \texttt{genTerm}\ \theta^\star\ (T^\ell) \wedge \kappa^\star(lab^{\theta^\star}(T^\ell), n) \wedge \\
 & & \texttt{genProc}\ n\ \theta^\star\ P \\
\quad \mid \texttt{genProc}\ n\ \theta^\star\ (\textbf{in}\ x.\, P) & = & (\forall\, \ell'.\, \forall\, m.\, \kappa^\star(\ell', m) \wedge \widehat{G}(m, n) \Rightarrow \\
 & & \quad \texttt{genProc}\ n\ (\theta^\star[x \mapsto \ell'])\ P) \\
\quad \mid \texttt{genProc}\ n\ \theta^\star\ (\textbf{store}\ T^\ell.\, P) & = & \texttt{genTerm}\ \theta^\star\ (T^\ell) \wedge \sigma^\star(lab^{\theta^\star}(T^\ell), n) \wedge \\
 & & \texttt{genProc}\ n\ \theta^\star\ P \\
\quad \mid \texttt{genProc}\ n\ \theta^\star\ (\textbf{read}\ x.\, P) & = & (\forall\, \ell'.\, \sigma^\star(\ell', n) \Rightarrow \texttt{genProc}\ n\ (\theta^\star[x \mapsto \ell'])\ P) \\
\quad \mid \texttt{genProc}\ n\ \theta^\star\ (\textbf{case}\ T^\ell\ \textbf{of}\ f(T_1^{\ell_1}, \ldots, T_j^{\ell_j}; x_{j+1}, \ldots, x_k)\ P_1\ \textbf{else}\ P_2) = & & \\
 & & \texttt{genTerm}\ \theta^\star\ (T^\ell) \wedge \bigwedge_{i=1}^{j} \texttt{genTerm}\ \theta^\star\ (T_i^{\ell_i}) \wedge \\
 & & (\forall\, \ell'_{j+1}, \ldots, \ell'_k.\, (\exists\, \ell', \ell'_1, \ldots, \ell'_j. \\
 & & \quad \gamma(\ell, \ell') \wedge f(\ell', \ell'_1, \ldots, \ell'_k) \wedge \\
 & & \quad \bigwedge_{i=1}^{j} \texttt{equal}\ \theta^\star\ (\ell'_i, T_i^{\ell_i})) \Rightarrow \\
 & & \quad\quad \texttt{genProc}\ n\ (\theta^\star[x_i \mapsto \ell'_i]_{i=j+1}^{k})\ P_1) \wedge \\
 & & \texttt{genProc}\ n\ \theta^\star\ P_2 \\
\quad \mid \texttt{genProc}\ n\ \theta^\star\ (P_1 \mid P_2) & = & \texttt{genProc}\ n\ \theta^\star\ P_1 \wedge \texttt{genProc}\ n\ \theta^\star\ P_2 \\
\quad \mid \texttt{genProc}\ n\ \theta^\star\ (!P) & = & \texttt{genProc}\ n\ \theta^\star\ P \\
\\
\texttt{fun genNet}\ (n[P, S]) & = & \texttt{genProc}\ n\ \theta^\star\ P \wedge \\
 & & (\forall\, T^\ell \in S.\, \texttt{genTerm}\ []\ (T^\ell) \wedge \sigma^\star(lab^{[]}(T^\ell), n)) \\
\quad \mid \texttt{genNet}\ (N_1 \parallel N_2) & = & \texttt{genNet}\ N_1 \wedge \texttt{genNet}\ N_2 \\
\\
\texttt{fun genGraph}\ \widehat{G} & = & \bigwedge_{(m,n) \in \widehat{G}} \widehat{G}(m, n) \\
\\
\texttt{fun genConstr}\ \widehat{G}\ N & = & \texttt{genGraph}\ \widehat{G} \wedge \texttt{genNet}\ N \wedge \texttt{genEta}\ N
\end{array}
$$

Where genEta N is defined in the text, and:

$$
\begin{array}{lcl}
\texttt{fun equal}\ \theta^\star\ (\ell', n^\ell) & = & \eta(\ell', \ell) \\
\quad \mid \texttt{equal}\ \theta^\star\ (\ell', x) & = & \eta(\ell', lab^{\theta^\star}(x)) \\
\quad \mid \texttt{equal}\ \theta^\star\ (\ell', f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k})) & = & \exists\, \ell'', \ell'_1, \ldots, \ell'_k. \\
 & & \quad \gamma(\ell', \ell'') \wedge f(\ell'', \ell'_1, \ldots, \ell'_k) \wedge \\
 & & \quad \bigwedge_{i=1}^{k} \texttt{equal}\ \theta^\star\ (\ell'_i, T_i^{\ell_i})
\end{array}
$$

Table 4.5: Constraint Generation Algorithm

only simple translations have to be performed in order to comply with the syntax of ALFP of §4.5.1: for instance, $n \in \gamma(\ell)$ is expressed by $\gamma(\ell, n)$; $\forall\, (\ell', m) \in \kappa^\star.\ cl$ by $\forall\, \ell', m.\ \kappa^\star(\ell', m) \Rightarrow cl$; and $f(\ell_1, \ldots, \ell_k) \in \gamma(\ell)$ by $\gamma(\ell, \ell) \wedge f(\ell, \ell_1, \ldots, \ell_k)$.

Using these translations, the constraint generation function genConstr $\widehat{G}$ N can be defined in pseudo SML code in Table 4.5. It takes an abstract network topology $\widehat{G}$

and a network N in Labelled CBS$^{\sharp}$ as arguments and produces a clause in ALFP. The generation function itself relies on three auxiliary functions: `genGraph` $\widehat{G}$ produces an ALFP representation of $\widehat{G}$; `genNet` N produces the actual constraints for N according to the specification of the finite analysis; and `genEta` N produces the specification of the auxiliary predicate η (an adaption and generalisation of the related definition in [26]), which is needed to express that the intersection of two languages is non-empty, as required in the specification of EQ-NAME and EQ-VAR:

$$
\begin{aligned}
\texttt{fun genEta}\ N\ =\ & (\forall\, \ell_1, \ell_2.\ \exists\, u.\ \gamma(\ell_1, u) \wedge \gamma(\ell_2, u) \Rightarrow \eta(\ell_1, \ell_2)) \wedge \\
& \textstyle\bigwedge_{f \in \mathit{funs}(N)} (\forall\, \ell_1, \ell_2.\ (\exists\, \ell'_1, \ell'_{1,0}, \ldots, \ell'_{1,\mathit{arity}(f)}.\ \exists\, \ell'_2, \ell'_{2,0}, \ldots, \ell'_{2,\mathit{arity}(f)}. \\
& \qquad f(\ell'_1, \ell'_{1,1}, \ldots, \ell'_{1,\mathit{arity}(f)}) \wedge f(\ell'_2, \ell'_{2,1}, \ldots, \ell'_{2,\mathit{arity}(f)}) \wedge \\
& \qquad \textstyle\bigwedge_{i=1}^{\mathit{arity}(f)} \eta(\ell'_{1,i}, \ell'_{2,i})) \Rightarrow \\
& \qquad \eta(\ell_1, \ell_2))
\end{aligned}
$$

It can be shown easily with Definition 4.16 that $\eta(\ell_1, \ell_2)$ holds if and only if $L(\gamma, \ell_1) \cap L(\gamma, \ell_2) \neq \emptyset$. Considering this and the syntactic changes mentioned above, it is clear that an interpretation of the ALFP clause produced by `genConstr` $\widehat{G}$ N (as computed for example by the Succinct Solver) yields an analysis estimate $(\kappa^\star, \sigma^\star, \gamma)$ such that $(\kappa^\star, \sigma^\star, \gamma) \models_{\widehat{G}} N$. Together with Theorem 4.21 this means that we can automatically compute a safe estimate for the original analysis.

4.5.5 Precision and Complexity

Program analyses [104] have in common that they use *approximation* in order to overcome the undecidability of the problem to determine any aspect of the functional behaviour of a program written in some Turing-complete language (Rice's theorem [72]). This means that the Control Flow Analysis is expected to produce possibly larger sets of terms than will ever be sent or stored during execution of a network, as a trade-off for the fact that the analysis is guaranteed to terminate (see page 81). It is part of the craft of writing program analyses to find a balance between keeping the imprecision of the analysis sufficiently small (in order for the analysis to be useful) and ensuring that its computation remains tractable. In this section, we discuss the trade-off between precision and computational complexity we have decided on for our analysis.

Precision. The precision of the Control Flow Analysis can best be demonstrated by comparing it with related approaches such as our own preliminary study [99] and the analysis of LySa in [22]. The approach in this dissertation distinguishes itself from these by the use of a *local* substitution environment which leads to a much improved precision of the analysis.

In order to illustrate the change in precision if a *global* variable environment were

to be used instead, we define an alternative analysis with the following new component:

$$\theta \ : \ \mathcal{X} \rightarrow \wp(\mathbf{T}) \quad \text{global variable environment}$$

The original local substitution is formally replaced by this global environment and the new process judgement is:

$$(\kappa, \sigma, \theta) \models_{\widehat{G},n} P \quad \text{variant judgement for processes}$$

Furthermore, an auxiliary judgement is introduced:

$$\theta \models T : \psi \quad \text{judgement for terms}$$

It expresses that "ψ is a valid estimate for T under θ", i.e. ψ contains all ground terms T might evaluate to under variable substitutions recorded in θ. The rules of the judgement for terms defined in Table 4.6.

VAR-NAME ψ is a valid estimate of name n under θ iff ψ contains n.

VAR-VAR ψ is a valid estimate of variable x iff all of the possible bindings $\theta(x)$ of x are contained in ψ.

VAR-FUN ψ is a valid estimate of function $f(T_1, \ldots, T_k)$ iff ψ contains $f(U_1, \ldots, U_k)$ for all estimates U_i of T_i.

We present three rules of the variant judgement for processes in Table 4.6. The remaining ones and the variant judgement for networks are straightforward or analogous.

VAR-OUT (κ, σ, ψ) is a valid analysis of process **out** $T.\,P$ iff the following holds: for all estimates U of T the pair (U, n) is contained in κ, and (κ, σ, ψ) is a valid analysis of process P.

VAR-IN For the analysis of the reception **in** $x.\,P$, all (U, m) recorded in κ have to be examined. If there is an edge $(m, n) \in E(\widehat{G})$, the possible binding of U to x has to be recorded in $\theta(x)$. Also, (κ, σ, ψ) has to be a valid analysis of process P.

VAR-CASE In the analysis of **case** T **of** $f(T_1, \ldots, T_j; x_{j+1}, \ldots, x_k)$ P_1 **else** P_2, the estimate ψ of T and the estimates $\psi_1, \ldots, \psi_j$ of $T_1, \ldots, T_j$ are computed. For all $f(U_1, \ldots, U_k) \in \psi$ it is checked whether the components $U_1, \ldots, U_j$, which are supposed to match with the respective T_i, are contained in the respective estimates ψ_i. If so, the remaining components $U_{j+1}, \ldots, U_k$ are recorded in $\theta(x_{j+1}), \ldots, \theta(x_k)$ and the analysis of P_1 is computed. Furthermore, the analysis of the second branch P_2 is always computed.

Judgement for Terms

VAR-NAME	$\theta \models n : \psi$	*iff*	$n \in \psi$
VAR-VAR	$\theta \models x : \psi$	*iff*	$\theta(x) \subseteq \psi$
VAR-FUN	$\theta \models f(T_1, \ldots, T_k) : \psi$	*iff*	$\bigwedge_{i=1}^{k} \theta \models T_i : \psi_i \wedge$ $(\forall\, U_1 \in \psi_1, \ldots, U_k \in \psi_k.$ $f(U_1, \ldots, U_k) \in \psi)$

Judgement for Processes

...

VAR-OUT	$(\kappa, \sigma, \theta) \models_{\widehat{G},n}$ **out** $T.P$	*iff*	$\theta \models T : \psi \wedge$ $(\forall\, U \in \psi.\ (U, n) \in \kappa) \wedge$ $(\kappa, \sigma, \theta) \models_{\widehat{G},n} P$
VAR-IN	$(\kappa, \sigma, \theta) \models_{\widehat{G},n}$ **in** $x.P$	*iff*	$(\forall\, (U, m) \in \kappa.\ (m, n) \in E(\widehat{G}) \Rightarrow$ $U \in \theta(x)) \wedge$ $(\kappa, \sigma, \theta) \models_{\widehat{G},n} P$

...

VAR-CASE	$(\kappa, \sigma, \theta) \models_{\widehat{G},n}$ **case** T **of** $f(T_1, \ldots, T_j; x_{j+1}, \ldots, x_k)\ P_1$ **else** P_2	*iff*	$\theta \models T : \psi \wedge$ $\bigwedge_{i=1}^{j} \theta \models T_i : \psi_i \wedge$ $(\forall\, f(U_1, \ldots, U_k) \in \psi.\ \bigwedge_{i=1}^{j} U_i \in \psi_i \Rightarrow$ $\bigwedge_{l=j+1}^{k} U_l \in \theta(x_l) \wedge$ $(\kappa, \sigma, \theta) \models_{\widehat{G},n} P_1) \wedge$ $(\kappa, \sigma, \theta) \models_{\widehat{G},n} P_2$

...

Table 4.6: Variant Analysis for CBS$^\sharp$

The difference between the original Control Flow Analysis and its variant is that the context in which the variables are used is lost in the variant, making the analysis less precise. This is demonstrated in the following example.

Example 4.22 Consider the following network, where n_1 and n_2 are connected to n_3:

$$
\begin{aligned}
N \stackrel{\text{def}}{=}\ & n_1[\textbf{out}\ \mathsf{Msg}(a, b), \varepsilon] \parallel n_2[\textbf{out}\ \mathsf{Msg}(c, d), \varepsilon] \parallel \\
& n_3[\textbf{in}\ x.\ \textbf{case}\ x\ \textbf{of}\ \mathsf{Msg}(; x_1, x_2)\ \textbf{out}\ \mathsf{Msg}(x_1, x_2)\ \textbf{else nil}, \varepsilon]
\end{aligned}
$$

The results of the original and variant analysis are as follows:

$$\{\mathsf{Msg}(a,b), \mathsf{Msg}(c,d)\} \in \psi(x)$$
$$\{a,c\} \in \psi(x_1), \quad \{b,d\} \in \psi(x_2)$$
$$\{\mathsf{Msg}(a,b)\} \in \kappa_{n_1}, \quad \{\mathsf{Msg}(c,d)\} \in \kappa_{n_2}$$
$$\{\ \mathsf{Msg}(a,b)\ , \mathsf{Msg}(a,d), \mathsf{Msg}(c,b), \mathsf{Msg}(c,d)\ \} \in \kappa_{n_3}$$

This means that the variant analysis takes $\mathsf{Msg}(a,d)$ and $\mathsf{Msg}(c,b)$ to be possible messages of n_3, because when analysing the sending statement in n_3 it is not taken into account that x_1 and x_2 occur in a common context $\mathsf{Msg}(;x_1,x_2)$. The original analysis is context-sensitive in this respect and establishes, more precisely, that $\mathsf{Msg}(a,b)$ and $\mathsf{Msg}(c,d)$ are the only possible messages to be sent by n_3. ⊠

Sometimes, the precision of an analysis of a model can not only be improved by changing the analysis but also by reformulating the modelling. For example, the node n_3 of Example 4.22 might have been defined more simply as follows, without changing the observable behaviour of the network:

$$n_3[\textbf{in}\ x.\ \textbf{out}\ x, \varepsilon]$$

Both the original and the variant analysis compute identical message cache κ for a network with updated node n_3. Analogously, we prefer in Chapter 6 to have flat term structures in order to improve precision: for instance, instead of the term $\mathsf{IP}(dst, next, \mathsf{RREP}(dst, dst_seq, src, hop_cnt))$ (compare §1.1.3) we write equivalently $\mathsf{IP_RREQ}(dst, next, dst, dst_seq, src, hop_cnt)$; otherwise even the original analysis might return for example the estimate $\mathsf{IP}(n_4, n_2, \mathsf{RREP}(n_3, 0, n_1, 0))$, which is imprecise because the first components of IP and RREP should be equal (they represent both dst).

While this and similar modelling heuristics sometimes help improving the precision on an on-the-fly basis, they cannot provide higher precision in all cases, and ensuring that two modellings are equivalent is not always simple, making such modifications a source of modelling mistakes. In any case, when analysing communication protocols, which deal extensively with building new messages out of previously received message components, such heuristics cannot suffice and the variant analysis is no longer a viable option.

Asymptotic Complexity. The very good precision of the Control Flow Analysis has to be paid for by an increased computational complexity. We first evaluate this increase by a simple worst-case analysis.

Proposition 2 of [107] provides the following result about estimating the worst-case complexity of computing an interpretation of an ALFP clause cl: if $\#\mathcal{U}$ denotes the size of the universe, $|cl|$ the size of the clause, r the maximal nesting depth of quantifiers in

the clause, and $\#\rho$ the sum of cardinalities of predicates $\rho(R)$, such an interpretation can be computed in time

$$\mathcal{O}(\#\rho + \#\mathcal{U}^r \cdot |cl|).$$

This complexity result is easily obtained by transforming cl into a clause with all variables instantiated in all possible ways so that it can be solved by classical solvers for Boolean equation systems.

We can apply this result to the following situation which describes a simplified view of the Constraint Generation Algorithm (compare Table 4.5):

$$\begin{aligned} cl &= node_1 \wedge \cdots \wedge node_l \\ node_i &= \forall\, \ell_{i,1,1}, \ldots, \ell_{i,1,k_{i,1}}.\, R_{i,1}(\ell_{i,1,1}, \ldots, \ell_{i,1,k_{i,1}}) \Rightarrow \ldots \\ &\qquad \Rightarrow \forall\, \ell_{i,j_i,1}, \ldots, \ell_{i,j_i,k_{i,j_i}}.\, R_{i,j_i}(\ell_{i,j_i,1}, \ldots, \ell_{i,j_i,k_{i,j_i}}) \Rightarrow \mathbf{1} \end{aligned}$$

Let $arity_{max}$ denote the maximal arity of predicates $R_{i,j}$, r_1 denote maximal nesting depth of implications in cl

$$r_1 = max\ \{j_1, \ldots, j_l\},$$

and r_2 the maximal number of quantifiers in any precondition

$$r_2 = max\ \{k_{1,1}, \ldots, k_{1,j_1}, \ldots, k_{l,1}, \ldots, k_{l,j_l}\} = arity_{max}.$$

With these terms and the above cited Proposition 2 it is clear that the worst-case complexity of computing an interpretation of cl is governed by a polynomial in $\#\mathcal{U}$ with the following degree:

$$r = r_1 \cdot r_2$$

In contrast, the variant analysis has a degree of $r = r_2$ because a variable $\ell_{i,j,k}$ is never used in relations $R_{i,j+1}, \ldots, R_{i,j_i}$: the sequence of r_1 implications could be formally decomposed into r_1 conjunctions of implications with nesting depth one [26].

Accurate Complexity Bounds. The Succinct Solver is not based on the simple implementation idea suggested by Proposition 2 of [107] and achieves much lower complexity bounds in benign cases. Therefore, in Theorem 2 of [107], a system to perform the exact complexity calculation for an input term of ALFP is presented. This system is based on the fundamental idea of counting the number of satisfying environments for preconditions to determine the number of necessary computation steps. For example, in the case of $R_{i,1}(\ell_{i,1,1}, \ldots, \ell_{i,1,k_{i,1}})$ there are $\#\rho(R_{i,1})$ satisfying environments because all $\ell_{i,1,1}, \ldots, \ell_{i,1,k_{i,1}}$ are all-quantified and can thus be unified with any tuple $\vec{a} \in \rho(R_{i,1})$. In the case of $R_{i,2}(\ell_{i,2,1}, \ldots, \ell_{i,2,k_{i,2}})$ one has to consider $\#\rho(R_1) \cdot \#\rho(R_2)$ environments resulting from the combination of environments found before and satisfactions for the newly introduced variables, and so on. Thus, the resulting complexity of solving the

clause $node_i$ can be described as follows:

$$\mathcal{O}(\#\rho(R_{i,1}) + \#\rho(R_{i,1}) \cdot \#\rho(R_{i,2}) + \ldots + \#\rho(R_{i,1}) \cdot \ldots \cdot \#\rho(R_{i,j_i}))$$

Note that by taking $\#\rho(R_{i,1}) \cdot \cdots \cdot \#\rho(R_{i,j_i})$ as the governing term and using the estimate $max_{i,j}\ \#\rho(R_{i,j}) \leq \#\mathcal{U}^{arity_{max}}$, we could obtain the worst-case polynomial $(\#\mathcal{U}^{arity_{max}})^{r_1}$ from this more precise bound.

Lowering the Complexity. The lower complexity bound suggests a way of addressing the problem that the terms $\#\rho(R_{i,1}) \cdot \cdots \cdot \#\rho(R_{i,j_i})$ can grow quite large and indicate a high complexity of the analysis: the fewer tuples the interpretations of $\rho(R_i)$ contain, the lower the complexity will be. This idea may seem to have little use in general, since one cannot normally influence the outcome of an analysis without changing its semantics. However, it turns out to be of great importance when considering the analysis of a Dolev-Yao attacker. For example, in the case studies considered in Chapter 6, honest nodes produce only very few tuples, the analysis of the attacker on the other hand frequently produces the maximum number of tuples $\#\mathcal{U}^{arity_{max}}$ for a predicate R with arity $arity_{max}$, since all attack possibilities have to be considered. In §5.1.3, we have therefore devised a type discipline which makes sure that the attacker produces only the terms which have any chance of being processed by honest nodes. We have found this technique to be very effective in keeping the security analyses of Chapter 6 tractable (see our practical results in Table 4.7).

On the other hand, several heuristics for keeping the size of the universe $\#\mathcal{U}$ small suggest themselves: for example, labelling of identical ground terms with identical labels and labelling of the attacker-produced terms with a single label. This is indeed the labelling we are using to compute the examples in Chapter 6. As arbitrary labellings are allowed in §4.5.2, the safety of the analysis is never put into question by applying such heuristics, but it has to be ensured that the precision remains good enough. We have implemented various such heuristics which are documented in the actual implementation.

Nonetheless, while these improvements have helped to produce an efficient analysis for the protocols modelled in this thesis, this does not mean that we have achieved scalability: for example, the exponential increase in worst-case complexity for protocols requiring higher arities of relations can easily overpower our heuristics and lead to unacceptable running times. Using the variant analysis with its lower complexity may help to analyse some more protocols (provided that the mentioned precision problems do not prevent that). Note however that the complexity increase of the variant analysis is exponential in the arities of relations as well (the exponent depends on r_2).

Network	#ns	att	ty	$\#\mathcal{U}$	$\#\rho$	$\#\rho(R_{max})$	r_1	$arity_{max}$	CPU time
Protocol 1	3	yes	yes	21	189	56	1	3	0.060s
Protocol 1	3	yes	no	21	214	85	1	3	0.132s
Protocol 2	3	yes	yes	32	336	88	3	4	0.236s
Protocol 2	3	yes	no	32	1017	548	3	4	5.716s
Protocol 3	3	yes	yes	37	805	344	5	5	3.220s
Protocol 3	3	yes	no	37	2974	1308	5	5	232.522s
Protocol 4	3	yes	yes	44	807	344	6	5	3.176s
Protocol 4	3	yes	no	44	2977	1308	6	5	1441.522s
μSAODV_RD_DS	3	no	no	61	478	48	7	6	0.156s
μSAODV_RD_DS	3	yes	yes	64	1875	888	7	6	32.878s
μSAODV_RD_HC	3	no	no	32	105	3	6	7	0.012s
μSAODV_RD_HC	3	yes	yes	36	345	138	6	7	12.829s
μSAODV_RM	5	no	no	82	256	6	4	5	0.020s
μSAODV_RM	5	yes	yes	83	1151	403	4	5	4.904s
μARAN_RD	3	no	no	69	321	3	12	7	0.024s
μARAN_RD	3	yes	yes	73	1189	533	12	7	1144.620s
μARAN_RM	3	no	no	54	94	0	8	6	0.008s
μARAN_RM	3	yes	yes	56	196	0	8	6	0.084s
μARAN_RM	3	yes	no	56	1707	1024	8	6	25.297s

Table 4.7: Running time measurements

Practical Results. Table 4.7 displays some measurements undertaken for the analyses of protocols described in §5.3 and Chapter 6, where the following abbreviations are used (in addition to those defined in the text):

#ns	Number of nodes
att	Presence of attacker
ty	Use of the type discipline of §5.1.3 for attacker analysis
$\#\rho(R_{max})$	Cardinality of the interpretation of the relation with greatest arity

A standard workstation (Dual Pentium IV, 3.2 GHz; 1 GB RAM) was used to compute the analyses. The results for Protocols 1–4 confirm that using the type discipline of §5.1.3 for the analysis of the attacker greatly reduces the analysis complexity. The influence of $\#\mathcal{U}$, r_1, and $arity_{max}$ on the amount of CPU time required becomes evident as well. In general, analyses not using the type discipline for attacker analysis may exhibit a high complexity, whereas analyses without attacker compute efficiently.

Chapter 5

Security of Mobile Ad-Hoc Networks

In Chapter 3 we have presented a calculus particularly suitable to model the behaviour of mobile ad-hoc networks and in Chapter 4 we have defined a Control Flow Analysis for this calculus in order to overapproximate the sets of terms transmitted and stored in a network. In this section we show how these two results can be combined to yield a framework for security analysis of mobile ad-hoc networks.

In general, formal verification and analysis frameworks consist of three parts addressing the following tasks [75]:

System modelling Using a *description formalism*, e.g. a process calculus, in order to define the system in question. In the case of security analysis, this task breaks down into two subtasks: modelling the honest system parts and modelling the attacker.

Property specification Using a *specification formalism*, e.g. a suitable logic, in order to define the property in question.

Analysis Using an *analysis method* in order to decide whether a system description satisfies a given property.

In terms of system modelling, we can clearly make use of CBS$^\sharp$ as a description formalism for modelling of mobile ad-hoc networks. However, it remains to be shown how to sensibly define the attacker in this setting. It turns out that it is necessary to deviate from the common Dolev-Yao attacker formulation which is too strong to be applicable to this problem. A parametric attacker is introduced which also gives a more fine-grained control over the security questions to be analysed. For property specification, we propose to use mediated equivalence. Finally, establishing the required equivalences automatically can be achieved by employing our Control Flow Analysis.

This chapter is in three sections. In §5.1, we introduce our attacker model and describe how it can be efficiently implemented. We present an important security property in the setting of mobile ad-hoc networks in §5.2. We apply the derived framework in §5.3, where an instructive sequence of small protocols is examined.

5.1 Attackers

Most systems for protocol analysis, e.g. [91, 51, 20], rely on what is commonly known as the *Dolev-Yao model of security* [44]. For our discussion in this section, it is helpful to disassemble this model into the following three assumptions:

Omnipresence The view that in order to prove a protocol correct, an attacker controlling all communication paths has to be considered. This point has been introduced by Needham and Schroeder in their seminal paper [101]:

> "We assume that an intruder can interpose a computer in all communication paths, and thus can alter or copy parts of messages, replay messages, or emit false material. While this may seem an extreme view, it is the only safe one when designing authentication protocols."

Perfect Cryptography This assumption states that ciphertext can only be

(1) decrypted if the decryption key is present, and

(2) generated using encryption with the appropriate encryption key and plaintext.

Adopting this convention permits to view basic cryptographic operations such as encryption and decryption as algebraic operators on atomic abstractions of names, keys, and so on. More complex sets of operators such as *xor* and finite-field exponentiation, with larger sets of algebraic laws, can be handled as well (see e.g. [92]).

Finite Set of Capabilities An idealised attacker is defined which has a finite set of capabilities at his disposal. It is assumed that this attacker is equivalent to one with an infinite set of capabilities which is however playing according to the rules of perfect cryptography (that it is justified to make this assumption can be confirmed formally, see for instance [33]).

Because of its tractability, the Dolev-Yao attacker has become the prevalent model for protocol analysis, especially with automated techniques.

In this section, we argue that in the setting of mobile ad-hoc networks it might be appropriate to deviate from the assumption of omnipresence and to assume instead that even the attacker has to comply with the locality principle introduced by the explicit

modelling of the network's topology. We develop this idea in §5.1.1. This naturally leads to the question of the strength of multiple attackers and attack scenarios which we investigate in §5.1.2. Finally, in §5.1.3, we address the issue of efficiently computing the behaviour of the attacker with our Control Flow Analysis.

5.1.1 The Parametric Attacker

Given the above quotation by Needham and Schroeder, one is tempted to assume that the Dolev-Yao attacker is the only valid model for *all* kinds of protocols, and not only for authentication protocols. This is because the view of being able to defend against an omnipresent attacker as "the only safe one" seems intuitively right. It comes thus as a surprise at first that the community concerned with developing secure mobile ad-hoc networks, e.g. [112, 61, 126], does not employ this model. Neither do they formally specify an alternative model, but it is apparent that the de-facto model used in their works is the following: the attacker is just another node in the network, restrained by the environment and playing by the rules of the perfect cryptography assumption; if there are multiple attackers, they can collude to perform stronger attacks.

It is clear that the de-facto model might seem more realistic to protocol developers, however is it justified to use it as a formal model? The main argument for answering this question positively is the fact that an unmodified Dolev-Yao attacker can compromise any network: this is because – in contrast to authentication and similar protocols – a network is to be regarded as compromised if no communication can take place (after all, the goal of a communication protocol is to establish communication in the first place). Were the attacker able to intercept any sent message as the Needham-Schroeder paradigm describes, this would be easy to achieve (by discarding any message).

This discussion shows that it is impossible to establish a view of an attacker in mobile ad-hoc networks which would be "safe" in the sense of Needham and Schroeder, since the omnipresent attacker is imaginable but cannot be defended against. We propose therefore to give up the assumption of omnipresence but retain perfect cryptography and the finite set of capabilities to yield a formal attacker model in the setting of mobile ad-hoc networks. We call this a *parametric attacker* as it carries the following parameters:

(1) Number of nodes owned by the attacker

(2) Initial amount of knowledge at the attacker's disposal

Furthermore, the nodes of the parametric attacker have to abide by the network topology just as regular nodes, thus making network connectivity an additional implicit parameter. This yields a very intuitive and realistic notion of an attacker, as the following example illustrates:

Example 5.1 Figure 5.1 shows a parametric attacker owning two nodes n_{A_1} and n_{A_2} which are connected to honest nodes n_1 and n_4, respectively. ⊠

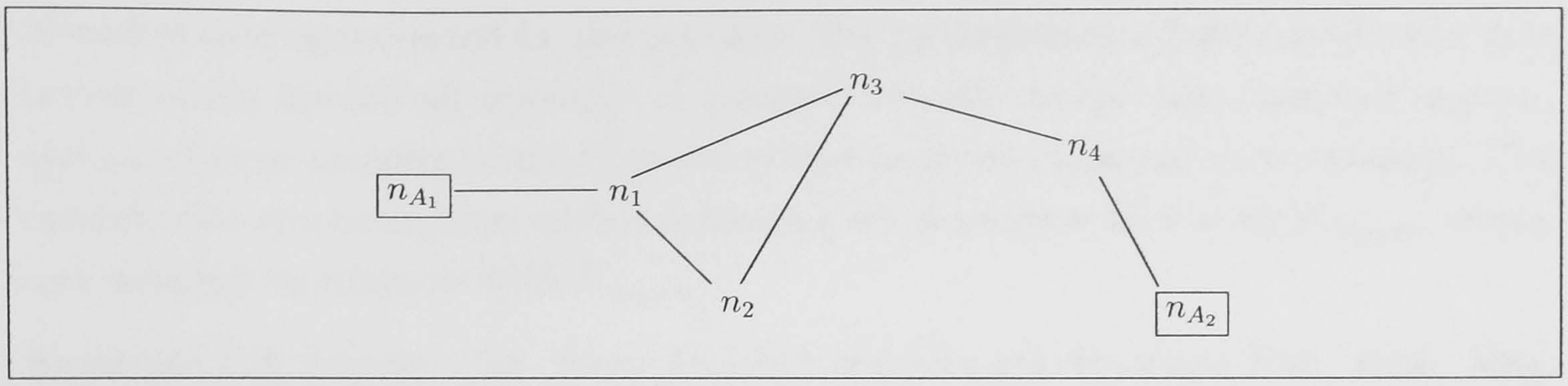

Figure 5.1: Example 5.1: parametric attacker

The challenge for the analyst is to decide which type of attacker a specific protocol should be resilient against. This also leads to the consideration of attack scenarios and the possibility of attacker hierarchies which are discussed in §5.1.2.

Formalisation. We formalise the parametric attacker within the framework of CBS$^\sharp$. As we give up the assumption of omnipresence from the Dolev-Yao model, this can simply be achieved by representing the attacker using one or more CBS$^\sharp$ nodes, but in addition we have to make sure that the assumptions of perfect cryptography and the finite set of capabilities are to be retained. Therefore, we first have to show that the perfect cryptography assumption can be adequately implemented in CBS$^\sharp$. It is important to note that – other than in the Spi Calculus [4], for instance – the algebraic laws of encryption and decryption are not hardcoded into the operational semantics of the calculus. It is thus easily possible (as it would be for example in CSP [124]) to write a process violating perfect cryptography, e.g. to read the contents of an encrypted message without the appropriate key. It is however possible to define a correct notion of the attacker using a simple partitioning of function symbols. This procedure is described in the following.

Assume the set of function symbols $\mathcal{F}$ is partitioned into the following subsets:

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

One- and *two-way functions* signify, as commonly found in cryptographic contexts, functions which are thought to be very difficult to reverse or are easily reversed, respectively. *Secret functions* are introduced for more specific modelling purposes: for example, in §3.4.2 we have introduced functions PrivKey and PubKey to yield key pairs for asymmetric encryption when applied to a seed. These functions in general may only be applied by certain protocol participants and therefore kept secret from the at-

tacker. *Message constructors* are functions which yield messages of valid format within the particular network in question. Conceptually, they are two-way functions, but we decide to introduce them at this point as a simple optimisation to reduce the number of useless messages created by the attacker. The optimisation is based on the fact that honest nodes discard all messages of incorrect format via the case construct anyway, and an efficient attacker would thus not bother to try to construct such messages. The optimisation can be undone, without changing the semantics, by leaving $\mathcal{F}_{message}$ empty and merging its contents with $\mathcal{F}_{twoway}$.

Example 5.2 Assume that some protocol employs the functions Pair, Hash, Msg, AsymEnc, PubKey, and PrivKey (with their obvious meanings). To specify the attacker properly, an adequate partitioning of this set of functions is given by:

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

⊠

$$\begin{aligned}
P_1(f) &\stackrel{\text{def}}{=} \textbf{!in}\ f(;x_1,\ldots,x_{\mathsf{arity}(f)}).\ \textbf{store}\ x_1.\ \ldots \textbf{store}\ x_{\mathsf{arity}(f)} \\
P_2(f) &\stackrel{\text{def}}{=} \textbf{!read}\ x_1.\ \ldots \textbf{read}\ x_{\mathsf{arity}(f)}.\ \textbf{out}\ f(x_1,\ldots,x_{\mathsf{arity}(f)}) \\
P_3(f) &\stackrel{\text{def}}{=} \textbf{!read}\ f(;x_1,\ldots,x_{\mathsf{arity}(f)})\ \textbf{store}\ x_1.\ \ldots \textbf{store}\ x_{\mathsf{arity}(f)}\ \textbf{else nil} \\
P_4(f) &\stackrel{\text{def}}{=} \textbf{!read}\ x_1.\ \ldots \textbf{read}\ x_{\mathsf{arity}(f)}.\ \textbf{store}\ f(x_1,\ldots,x_{\mathsf{arity}(f)}) \\
P_5 &\stackrel{\text{def}}{=} \textbf{!read}\ x_1.\ \textbf{read}\ x_2.\ \textbf{symdec}\ x_1\ x_2\ x\ \textbf{store}\ x\ \textbf{else nil} \\
&\mid \textbf{!read}\ x_1.\ \textbf{read}\ x_2.\ \textbf{asymdec}\ x_1\ x_2\ x\ \textbf{store}\ x\ \textbf{else nil} \\
P_A &\stackrel{\text{def}}{=} (|_{f\in\mathcal{F}_{message}}\ P_1(f) \mid P_2(f)) \\
&\mid (|_{f\in\mathcal{F}_{twoway}}\ P_3(f)) \\
&\mid (|_{f\in\mathcal{F}_{oneway}\cup\mathcal{F}_{twoway}}\ P_4(f)) \\
&\mid P_5 \\
A(k_A, S_A) &\stackrel{\text{def}}{=} \|_{i=1,\ldots,k}\ n_{A_i}[P_A, S_A]
\end{aligned}$$

Table 5.1: The parametric attacker

The generic attacker process P_A and the parametric attacker $A(k, S_A)$ are defined in Table 5.1. CBS$^\sharp$'s explicit notion of store makes for a very straightforward and intuitive modelling of the attackers capabilities as processes, which seems more cumbersome in other approaches (e.g., [20]). P_A is composed of five processes describing the following Dolev-Yao capabilities:

(1) Receiving messages of valid formats and adding their components to the store

(2) Sending messages of valid formats constructed from items in the store

(3) Adding arguments of two-way functions to the store

(4) Applying one- and two-way functions to arguments from the store

(5) Performing symmetric and asymmetric decryption provided that the appropriate keys are in the store

To summarise, the attacker $A(k, S_A)$ carries the following parameters:

(1) k, the number of nodes the attacker owns

(2) S_A, the store containing the attacker's initial knowledge, distributed to all owned nodes

The network topology $\mathcal{T}$ may be seen as an implicit parameter.

Notation 5.3 In our implementation (see §4.5), we use the notation

$$\textsf{attacker}\ k\ [T_1, \ldots, T_i]$$

in order to specify an attacker $A(k, S_A)$ with $S_A = T_1, \ldots, T_k$. ⊠

The network topology $\mathcal{T}$ can be used to fine-tune the connectivity of the attacker. Of course, the classical Dolev-Yao attacker can be achieved as a boundary case, for instance by specifying a network topology which is a star topology with the node $A(1, S_A)$ at the centre and connections to n honest nodes. Two different ways of specifying a Dolev-Yao parametrisation of the attacker are shown in Figure 5.2.

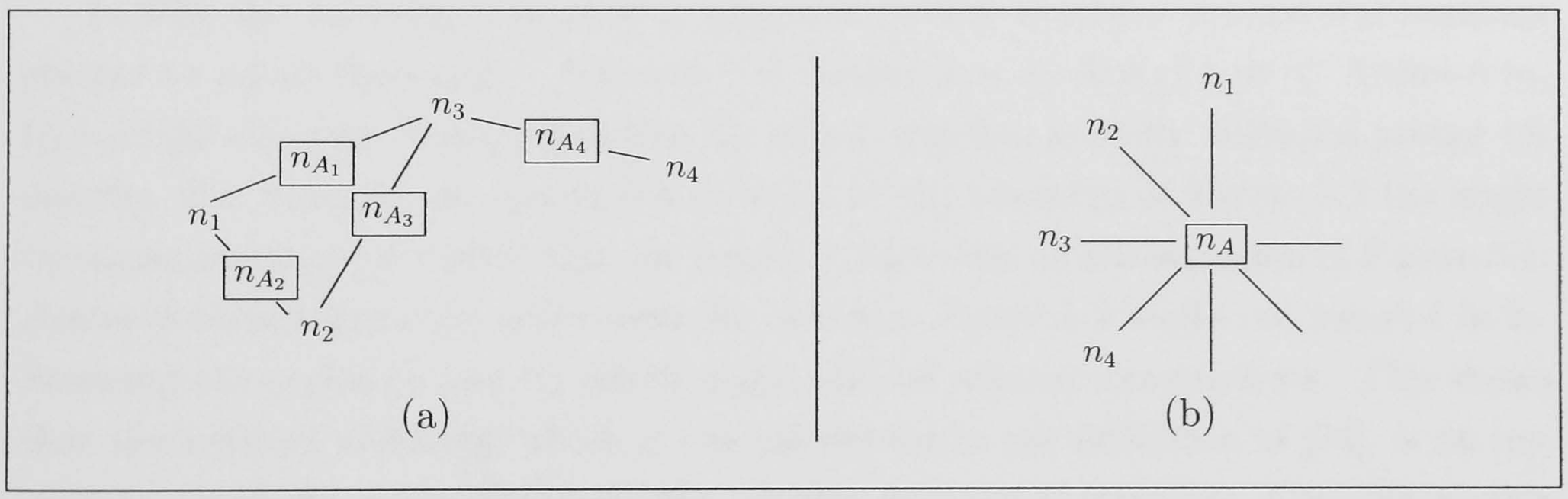

Figure 5.2: The Dolev-Yao attacker as a boundary case of the parametric attacker

5.1.2 Attack Scenarios

In the previous section we have defined a parametric attacker in order to relax the assumptions of the Dolev-Yao model for mobile ad-hoc networks. We have furthermore argued that a notion of "safety" against an attacker cannot exist in this setting since we can always imagine network topologies and respective attack formations which cannot be defended against. Hence, it is the task of the analyst to choose parameters for the attacker which are adequate for a given scenario.

Example 5.4 We present two scenarios and related possible parameter choices:

(1) An attacker captures a honest node and uses its cryptographic information (keys), but is not believed to have access to other nodes with whom he could jointly launch an attack. Parameter choice: $k_A = 1$ and $key \in S_A$, where key is a valid cryptographic key in the network.

(2) A group of attackers colludes to compromise a network but is not believed to be able to capture honest nodes and reveal cryptographic keys by direct physical access. Parameter choice: $k_A \geq 2$ and $S_A = \varepsilon$. ⊠

In order to choose scenarios, it would be helpful to judge the relative strength of attackers. Hu, Perrig, and Johnson [74] for example have devised an attacker hierarchy for mobile ad-hoc networks (see §2.3.4). They informally introduce the notation *Active-n-m* for an attacker who compromises n nodes (i.e., adds their valid keys to his knowledge) and owns a total number of m nodes. This attacker can be expressed formally using the parametric attacker $A(k_A, S_A)$ of Table 5.1:

Definition 5.5 (The Active-*n*-*m* attacker) For natural numbers m and n, let a network N be given, and let $key_1, \ldots, key_n$ be n different symmetric or private asymmetric keys, valid in the context of N. The *Active-n-m attacker* for N is given by:

$$\textit{Active-n-m} \stackrel{\text{def}}{=} A(m, key_1 \cup \cdots \cup key_n)$$ ⊠

In [74], the following hierarchy is proposed (where m and n are natural numbers greater or equal than one): *Active*-0-1 $\leq$ *Active*-0-m $\leq$ *Active*-1-m $\leq$ *Active-n-m*. One might challenge this proposition by asking whether it really defines a proper hierarchy. For example, an *Active*-0-4 attacker in the situation of Figure 5.2 (a) might be considered more powerful than an *Active*-1-2 attacker in the situation of Figure 5.1: *Active*-0-4 can silence the entire network, whereas *Active*-1-2 might not succeed in influencing the nodes n_2 and n_3 which might still be able to communicate. This shows that the network topology, which is not mentioned in the definition of [74], is an important parameter when discussing the relative strength of attackers. Considering this and the fact that the "attacker strength" seems impossible to define without reference to a specific protocol, we believe that the question of attacker orderings cannot be satisfactorily discussed in general but only when given a specific scenario.

5.1.3 A Type Discipline for Attacker Implementations

One of the key challenges when developing automated techniques for protocol analysis in a security context is to ensure that the computation of the analysis result remains tractable in spite of the overwhelming number of behavioural patterns exhibited by the attacker. Even with the Dolev-Yao assumption of a finite set of attacker capabilities

in place, model checking approaches for example have to face a huge branching factor introduced into the search space because the attacker may send one of infinitely many messages at any point in time; this issue can be mitigated for example by the use of symbolic techniques [10]. By comparison, our control flow analysis approach seems to fare better because it abstracts from a timeline, and we are also able to finitely represent an infinite set of terms by using tree grammars. On the other hand, the good precision of our analysis entails that the analysis of the attacker will nonetheless produce a large set of distinct terms, its size depending primarily on the size of the universe of labels used and the arity of the occurring functions, slowing down the analysis (see §4.5.5). In this section, we address this issue by presenting a type discipline to reduce the number of terms that have to be analysed.

The idea behind this type discipline is an extension of the simple optimisation presented in §5.1.1 which was based on the fact that honest nodes process only messages of correct format: we defined an attacker which produces only such messages. In §5.1.1, the notion of "message format" was determined only by checking that the outermost applied function is a valid message constructor; now it will be determined by a typing of all occurring terms, which again greatly reduces the number of futile terms produced by the attacker.

Judgement for Terms

$$
\begin{array}{rl}
\textsc{Ty-Name} & (E,\gamma) \vDash^{\theta^\star} n^\ell \ \ \textit{iff}\ n \in \gamma(\ell) \wedge \ell : \tau_n \in E \\
\textsc{Ty-Var} & (E,\gamma) \vDash^{\theta^\star} x \\
\textsc{Ty-Fun} & (E,\gamma) \vDash^{\theta^\star} f^\ell(T_1^{\ell_1},\ldots,T_k^{\ell_k}) \\
& \quad \textit{iff}\ \bigwedge_{i=1}^k (E,\gamma) \vDash^{\theta^\star} T_i^{\ell_i} \wedge \\
& \qquad (\bigwedge_{i=1}^k E \vdash \mathrm{lab}^{\theta^\star}(T_1^{\ell_i}) : \tau_i \wedge \mathit{TDecl}(f) = \tau_1 \times \ldots \times \tau_k \rightarrow \tau \ \Rightarrow \\
& \qquad\quad f(\mathrm{lab}^{\theta^\star}(T_1^{\ell_1}),\ldots,\mathrm{lab}^{\theta^\star}(T_k^{\ell_k})) \in \gamma(\ell) \wedge \ell : \tau \in E) \\
& \qquad\qquad \ldots
\end{array}
$$

Table 5.2: Typed Finite Analysis for Labelled CBS$^\sharp$

Let **Ty** denote a finite set of *types*, and let τ_n denote the base type of names. We assume further that for all $f \in \mathcal{F}$ with $\mathit{arity}(f) = k$ a unique *type declaration* $f : \tau_1 \times \cdots \times \tau_k \rightarrow \tau$ for f is defined, and the operator *TDecl* yields these function types, i.e. $\mathit{TDecl}(f) = \tau_1 \times \ldots \times \tau_k \rightarrow \tau$.

Notation 5.6 In our implementation (see §4.5), we use the notation

$$\textbf{signature}\ f(\tau_1,\ldots,\tau_k) : \tau\ [\textbf{oneway} \mid \textbf{twoway} \mid \textbf{secret} \mid \textbf{message}]$$

in order to express that function f has type declaration $\mathit{TDecl}(f) = \tau_1 \times \ldots \times \tau_k \rightarrow \tau$ and belongs to the function partition $\mathcal{F}_{oneway}$, $\mathcal{F}_{twoway}$, $\mathcal{F}_{secret}$, or $\mathcal{F}_{message}$ (see §5.1.1). ⊠

In order to make the type discipline work directly for the finite analysis of Table 4.5.2, we are concerned with types of labels, which are associated with sets of terms via the grammar γ. We add the following environment:

$$E \subseteq \mathcal{L} \times \mathbf{Ty} \quad \text{typing environment}$$

We write $\ell : \tau \in E$ to signify that label ℓ has type τ in this typing environment. E is called well-formed if every label is assigned at most one type. The judgement $E \vdash \ell : \tau$ means that ℓ is a label of type τ in a well-formed environment E:

$$\frac{E \text{ well-formed} \qquad \ell : \tau \in E}{E \vdash \ell : \tau}$$

The typed analysis is derived as follows: the rules FIN-NAME, FIN-VAR, and FIN-FUN of the judgement for terms in Table 4.4 are replaced by the three rules TY-NAME, TY-VAR, and TY-FUN in Table 5.2; judgement $\gamma \models^{\theta^\star} T^\ell$ is replaced by $(E, \gamma) \models^{\theta^\star} T^\ell$ in all rules of Table 4.4; and all other judgements there are formally extended with the component E. The new rules of the judgement for terms can be described as follows:

TY-NAME Similar to FIN-NAME, but additionally the typing environment is enlarged by the fact that ℓ is of type τ_n.

TY-VAR Analogous to FIN-VAR.

TY-FUN Similar to FIN-FUN, but the rule $\ell \rightarrow f(lab^{\theta^\star}(T_1^{\ell_1}), \ldots, lab^{\theta^\star}(T_k^{\ell_k}))$ is added to γ only in the case that components $lab^{\theta^\star}(T_1^{\ell_i})$ of f are of type τ_i, for $i = 1, \ldots, k$ as prescribed by the type declaration $TDecl(f)$. If this is the case, the label ℓ of the resulting term is added to E with type τ.

The typed finite analysis can again be straightforwardly converted into a constraint generation algorithm by replacing the appropriate statements in Table 4.5 by the ones of Table 5.3. Note that $\ell : \tau \in E$ is expressed by the truth of the statement $\tau(\ell)$, where all types τ are reinterpreted as predicates in ALFP.

$$\begin{array}{lll}
\texttt{fun genTerm}\ \theta^\star\ (n^\ell) & = & \gamma(\ell, n) \wedge \tau_n(\ell) \\
\quad \mid \texttt{genTerm}\ \theta^\star\ x & = & \mathbf{1} \\
\quad \mid \texttt{genTerm}\ \theta^\star\ (f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k})) & = & \bigwedge_{i=1}^k \texttt{genTerm}\ \theta^\star\ (T_i^{\ell_i}) \wedge \\
 & & (\bigwedge_{i=1}^k \tau_i(\ell_i) \ \Rightarrow \\
 & & \quad \gamma(\ell, \ell) \wedge f(\ell, lab^{\theta^\star}(T_1^{\ell_1}), \ldots, lab^{\theta^\star}(T_k^{\ell_k})) \wedge \tau(\ell)). \\
 & & \text{where } TDecl(f) = \tau_1 \times \ldots \times \tau_k \rightarrow \tau
\end{array}$$

Table 5.3: Typed Constraint Generation Algorithm

The following example gives an impression of the benefit of applying the type discipline in practice:

Example 5.7 Assume we compute the analysis result of a parametric attacker n_A (Table 5.1) with the following subprocess:

$$P_4(\mathsf{Cert}) \stackrel{\text{def}}{=} \textbf{!read}\ x_1. \ldots \textbf{read}\ x_3. \textbf{store}\ \mathsf{Cert}(x_1, \ldots, x_3)$$

Furthermore, $TDecl(\mathsf{Cert}) = \tau_n \times \tau_{\mathsf{PubKey}} \times \tau_{\mathsf{Sign}} \rightarrow \tau_{\mathsf{Cert}}$, and Cert describes a digital certificate which binds a name to a public key, and this binding is asserted by a signature of the two. Assume that – with or without type discipline – at the end of the analysis the set $\{\ell : (\ell, n_A) \in \sigma^\star\}$ contains $\#all$ elements, but, when considering types, there are only $\#name$, #PubKey, and #Sign elements which are typed τ_n, τ_{PubKey}, and τ_{Sign}, respectively. We want to compute the cardinality of the set $\gamma(\ell)$ where ℓ is the label assigned by the labelling function to Cert in $P_4(\mathsf{Cert})$. By application of rules FIN-REPL, FIN-READ, and FIN-STORE, the judgement $(\kappa^\star, \sigma^\star, \gamma) \models^{[]}_{\widehat{G},n} P_4(\mathsf{Cert})$ is equivalent to:

$$\begin{array}{c} \forall\ (\ell'_3, n_A) \in \sigma^\star.\ \forall\ (\ell'_2, n_A) \in \sigma^\star.\ \forall\ (\ell'_1, n_A) \in \sigma^\star. \\ \gamma \models^{[x_i \mapsto \ell'_i]_{i=1}^3} \mathsf{Cert}^\ell(x_1, \ldots, x_3) \wedge (\ell, n) \in \sigma^\star \end{array}$$

If FIN-FUN is applied, it is therefore clear that $|\gamma(\ell)| = \#all^3$, application of TY-FUN on the other hand yields $|\gamma(\ell)| = \#name \cdot \#\mathsf{PubKey} \cdot \#\mathsf{Sign}$ elements. The fact that in general

$$\#name \cdot \#\mathsf{PubKey} \cdot \#\mathsf{Sign} \ll \#all^3$$

shows the benefit of a typed analysis. For example, in a network of five honest nodes one might have $\#all = 5 \cdot 10$ and $\#name = \#\mathsf{PubKey} = \#\mathsf{Sign} = 5$, therefore the difference between the two cardinalities would be three orders of magnitude. ⊠

If we were to integrate this type discipline fully into our framework, Theorem 4.21 would have to be proved once again using the new rules. Unfortunately, this is not possible by simple adaption of the proof: take for example an analysis of the parametric attacker using the new rules. In general, this analysis produces a grammar γ which contains strictly less elements than when produced with the untyped analysis. Thus there will be sets of terms produced by γ which are no longer safe with respect to the corresponding sets of the analysis of Table 4.1. In order to establish results for the typed analysis similar to Theorem 4.21, it would thus be necessary to embrace the type discipline in full: the analysis of Table 4.1 would have to be typed, and ultimately (for the proof of Theorem 4.9) CBS$^\sharp$ would have to be enriched with types as well. The required changes, together with the implied loss of presentational simplicity, turned out to be too extensive to be included in this dissertation. However, we think that it would be worthwhile to include them in further developments of our framework or in

other control flow analysis frameworks which consider attackers and rich term structure, since it considerably reduces the complexity of computing these analyses (see §4.5.5). Note however the implications on possible implementations of a model analysed in this fashion: run-time checks which can confirm the type of received message components might be necessary.

For the sake of this dissertation, we can still use the type discipline in the following way: given the missing theorems, the typed implementation cannot be used to establish results which require safety of the analysis, e.g. to prove protocols correct – the untyped implementation has to be used instead; it is however possible to use a typed analysis of the attacker in conjunction with an untyped analysis of the honest part of the network *solely for the purpose of finding flaws.* This approach is viable, since every term produced by the typed attacker which induces a flaw will also be produced by the untyped attacker.

5.2 A Consistency Condition for Routing Networks

As discussed in §1.1.4, many non-formal works on security analysis of specific systems are attack-driven: a number of known general attacks are considered and it is argued why the system in question is safeguarded against these; once a new kind of attack is discovered, the system has to be reevaluated and potentially patched in order to defend against it. We have argued that this approach seems unsatisfactory because it relies on the current, inherently limited knowledge about potential attacks: complete security can never be achieved in this way. The formal methods community has therefore argued for many years to tackle the problem from a different perspective. Instead of considering attacks, the absolute security requirements of the system are investigated. Given a formal model of the system and a formalisation of the security requirements as a desirable property, one can then attempt to prove the security of the system using some verification framework. On success, it is certain to be safeguarded against *all* possible attacks which try to violate the property, provided of course that these attacks are expressible in the chosen formalisation of the setting.

However, depending on the system in question, defining the right security property can be challenging. In classical point-to-point security protocols this problem is simplified by the fact that these protocols are *designed* to achieve one or more specific security properties (thus, there are protocols *for* authentication, *for* secrecy, etc.). In contrast, communication protocols provide complex network services which are per se unrelated to any of the common security properties. As a starting point for defining suitable security properties in this setting, we propose the following statement as a guideline: securing a communication protocol means to ensure that it can provide its network services unconditionally, even in an adversarial environment. In the remainder of this section, a property for routing protocols is developed in this manner (§5.2.1).

In §5.2.2 we show how to use our current modelling and analysis framework to check this property automatically.

5.2.1 Topology Consistency

Following the guideline stated above, finding a security property for communication protocols has to start with determining the network service to be provided. In the case of routing protocols, the service consists in establishing paths between communication partners in the network. We require found paths to be *valid*, i.e. that they exist in the network and contain only accepted protocol participants. As found paths are represented in routing tables and validity can only be verified with knowledge of the network topology, it is clear that the appropriate property for this scenario is based on the tense relation between virtual (stores S) and actual (network topology $\mathcal{T}$) representation of network connectivity.

Furthermore, the network service has to be provided even in an adversarial environment of some sort. We have argued in §5.1.2 that this environment can be modelled at the discretion of the analyst with an appropriate parametric attacker. In order to understand how such an attacker can compromise the validity of a path, it is helpful to remind oneself that every honest 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 of correct format. 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, where a piece of information is to be considered malicious if it violates the validity of a found path. These considerations provide the motivation for the following definition.

We use mediated equivalence (see §3.3) in order to specify a consistency condition for routing networks. This condition is explicitly parametric in a mediator expressing the validity of a routing path:

Definition 5.8 (Consistency Mediator) Let $\widehat{G}$ be an abstract network topology, and let *dst*, *next*, and *dist* be mappings which yield destination, next hop, and number of hops to the destination, when applied to a term U denoting a routing table entry holding this information. The mediator $\mu_n^{\widehat{G}}$ is called a *consistency mediator* if it is simple and the following holds:

$$\mu_n^{\widehat{G}}(U) = \varepsilon \quad \textit{iff} \quad \text{there exists a path } n_1, \ldots, n_k \text{ in } \widehat{G} \text{ such that}$$

$$\begin{array}{llllll} (1) & n_1 = n & & & & \\ (2) & \text{if} & dst(U) & \text{is defined then} & n_k & = dst(U) \\ (3) & \text{if} & next(U) & \text{is defined then} & n_2 & = next(U) \\ (4) & \text{if} & dist(U) & \text{is defined then} & k-1 & = dist(U) \end{array}$$

⊠

This definition of the consistency mediator says intuitively that if a routing table entry U is stored at n, the network topology should allow for a path from n to the destination via the next hop, and that the distance information (if available) should be accurate. Note that the definition makes use of three mappings from U to components of U because the format for routing table entries may vary. The following is an example of a concrete consistency mediator:

Example 5.9 We assume a protocol for hop-by-hop routing which uses the notation $\mathsf{Rte}(n_{dst}, n_{next})$ in order to represent a routing table entry, meaning that packets addressed to n_{dst} arriving at the current node n will be forwarded to n_{next} as the next hop. Then it is easy to see that $\mu_n^{\widehat{G}}$ defines a consistency mediator in the context of this protocol:

$$\mu_n^{\widehat{G}}(U) = \begin{cases} \varepsilon & \text{if for } U = \mathsf{Rte}(n_{dst}, n_{next}) \text{ there exist locations} \\ & n_1 = n, n_2 = n_{next}, \ldots, n_k = n_{dst} \text{ such that} \\ & (n_i, n_{i+1}) \in \widehat{G} \text{ for all } i \in \{1, \ldots, k-1\} \\ U & \text{otherwise} \end{cases}$$

⊠

We can now proceed to define *topology consistency*, a central notion in this dissertation: it uses CBS$^\sharp$ to formalise a notion of security for mobile ad-hoc networks which can be readily checked by the Control Flow Analysis. Informally, it expresses that a network finds only valid topological information even if an attacker is present.

Definition 5.10 (Topology Consistency) Let N be a network of routers and let the following parameters be defined:

(TC1) A parametric attacker $A(k_A, S_A)$.

(TC2) A network topology $\mathcal{T}$, admissible on $N \parallel A(k_A, S_A)$.

(TC3) A consistency mediator $\mu_n^{\mathcal{G}(\mathcal{T})}$.

N is said to be *topology consistent under* (TC1)–(TC3) if the following equivalence holds:

$$N \simeq_{\mathcal{T}}^{(\mu_n^{\mathcal{G}(\mathcal{T})})_{n \in \mathcal{N}}} N \parallel A(k_A, S_A)$$

⊠

The equivalence of Definition 5.10 describes intuitively that "during evolution of network $N \parallel A(k_A, S_A)$, the stores of nodes in N can contain only routing table entries that are either valid with respect to $\mathcal{G}(\mathcal{T})$ or can be created during evolution of N alone as well (and vice versa)". This is because from the definition of mediated equivalence we know that $A(k_A, S_A)$ is allowed to store anything since the convergence predicate is only checked for all $n \in V(N) \cap V(N \parallel A(k_A, S_A)) = V(N)$. However, if interaction of N with $A(k_A, S_A)$ causes some information U to be stored by nodes of N, the equivalence cannot be established unless N would be able to store $\mu_n^{\mathcal{G}(\mathcal{T})}(U)$ as well. If U

describes valid information then $\mu_n^{\mathcal{G}(\mathcal{T})}(U) = \varepsilon$ which N can trivially store; otherwise $\mu_n^{\mathcal{G}(\mathcal{T})}(U) = U$ which is not normally stored by N (except there is a functional flaw in the protocol anyway).

The following two remarks help clarify the notion of security we provide with the above definition. Firstly, note that in CBS$^\sharp$ we do not allow nondeterministic message loss: the operational semantics forces all nodes to receive a message if they are able to receive it. This means that attacks which are based on message loss cannot be handled by this definition. Instead we provide a safety property, namely that the attacker can never succeed to insert certain terms into one of the stores of the honest nodes. For our application, this seems to be the right notion, since mobile ad-hoc networks are designed in particular to handle message loss due to changing environment conditions, and it is thus reasonable to believe that message loss cannot be exploited for an attack. This is not the case for other networking paradigms, a fact which has to be taken care of when trying to transfer our approach to other applications.

Secondly, mediated equivalence is certainly not the only way of expressing the kind of security property we are after, nor even the most straightforward one. For example, in a variant approach, we could have omitted behavioural equivalences altogether and define the property with the help of the control flow analysis only. However, we find that mediated equivalence provides a very elegant and also intuitive notion of the desired security property: by comparing a network of honest nodes N with $N \parallel M$, i.e. the same network extended by some a priori unknown network M, we are able to express the view that M might either be a new network of honest nodes, or of attackers. We believe that it might be hard to elegantly express this notion using other approaches.

In §5.3 we present examples of how the application of the topology condition works. Before we can do this, we have to determine in the next section how the Control Flow Analysis can help establishing the equivalence.

5.2.2 Automated Consistency Checking

The equivalence of Definition 5.10 may be proved by hand, which can however be a laborious and error-prone procedure. Instead we would like to use the Control Flow Analysis to establish the result. This can be achieved under certain conditions via the following theorem:

Theorem 5.11 *The following algorithm yields safe results about a network N (i.e. it never wrongly determines topology consistency of N):*

1. *Choose conditions (TC1)–(TC3) of Definition 5.10*
2. *With $A(k_A, S_A)$ as in (TC1) and $\mathcal{T}$ as in (TC2), compute:*

$$(\kappa^\star, \sigma^\star, \gamma) \models_{\mathcal{G}(\mathcal{T})} \textbf{labelling}(N \parallel A(k_A, S_A))$$

3. *With* $(\mu_n^{\mathcal{G}(\mathcal{T})})_{n\in\mathcal{N}}$ *as in (TC3), determine for each* $(\ell, n) \in \sigma^\star$ *with* $n \in V(N)$ *whether the implication*

$$\text{if } U \in L(\gamma, \ell) \text{ then } \mu_n^{\mathcal{G}(\mathcal{T})}(U) = \varepsilon$$

holds; if not, output "Unable to determine topology consistency" and terminate

4. *Output "Network* N *is topology consistent under (TC1)–(TC3)"*

Proof. A direct consequence of Corollary 4.15 and Theorem 4.21. □

The algorithm of Theorem 5.11 is guaranteed to terminate if all sets $L(\gamma, \ell)$ occurring in step 3 are finite. If it terminates with the message "Network N is topology consistent under (TC1)–(TC3)", N is proved to be topology consistent. If it terminates with "Unable to determine topology consistency", network N may or may not be topology consistent. In this case, the terms U with $\mu_n^{\mathcal{G}(\mathcal{T})}(U) \neq \varepsilon$ can be examined in order to determine whether they are indications of an attack or false positives. The following theorem helps establishing topology inconsistency of a network:

Theorem 5.12 *If* $(N \parallel M) \Downarrow_n^{\mathcal{T}} U$, $(\kappa^\star, \sigma^\star, \gamma) \vDash_{\mathcal{G}(\mathcal{T})} N$, $\mu_n^{\mathcal{G}(\mathcal{T})}(U) \neq \varepsilon$, *and* $\mu_n^{\mathcal{G}(\mathcal{T})}(U) \notin L(\gamma, \ell)$ *for all* $(\ell, n) \in \sigma^\star$, *then* $N \not\approx_{\mathcal{T}}^{(\mu_n^{\mathcal{G}(\mathcal{T})})_{n\in\mathcal{N}}} (N \parallel M)$.

Proof. By definition of mediated equivalence, it suffices to show that:

$$(N \parallel M) \Downarrow_n^{\mathcal{T}} U \text{ and } N \not\Downarrow_n^{\mathcal{T}} \mu_n^{\mathcal{G}(\mathcal{T})}(U) \text{ for some } U \text{ and } n \in V(N)$$

We have $(N \parallel M) \Downarrow_n^{\mathcal{T}} U$ by assumption. Since $\mu_n^{\mathcal{G}(\mathcal{T})}(U) \notin L(\gamma, \ell)$ for all $(\ell, n) \in \sigma^\star$, we have $(\mu_n^{\mathcal{G}(\mathcal{T})}(U), n) \notin \sigma$ $(*)$ with Theorem 4.21. We can apply Corollary 4.13 to $\mu_n^{\mathcal{G}(\mathcal{T})}(U) \neq \varepsilon$, $(*)$, and $(\kappa^\star, \sigma^\star, \gamma) \vDash_{\mathcal{G}(\mathcal{T})} N$ to have $N \not\Downarrow_n^{\mathcal{T}} \mu_n^{\mathcal{G}(\mathcal{T})}(U)$. □

In order to prove that N is not topology consistent under conditions (TC1)–(TC3), it thus suffices to provide an attacker M such that $(N \parallel M) \Downarrow_n^{\mathcal{T}} U$, and show in addition that network N alone is topology consistent.

5.3 Examples: Security Analysis

Small examples are usually preferred to large case studies (such as the ones in Chapter 6) when it comes to demonstrating and understanding basic concepts. However, there is no such thing as a "Needham-Schroeder public key protocol"-equivalent for routing protocols in mobile ad-hoc networks. In this section, we thus try to find some middle ground and present a series of "toy protocols" which are unrealistic, but share important similarities with their real-world counterparts to make for an interesting discussion of good and bad design for secure routing. In §5.3.1 we first present a series of four simple

routing protocols describing a route discovery step. The protocols incrementally build upon each other and are evaluated using an $A(1, n_A)$ attacker as single adversarial scenario. In §5.3.2 we consider alternative instantiations of the parametric attacker.

5.3.1 Four Toy Protocols

The protocols analysed in this section perform route discovery in the fashion of hop-by-hop routing protocols. In particular, they have in common that they use the notation $\mathsf{Rte}(n_{dst}, n_{next})$ of Example 5.9 in order to represent a routing table entry. Only the routing functionality is modelled, but the existence of additional protocol parts for the forwarding of data packets can be assumed for the discussion of possible attacks.

The choice of the number of nodes and the initialisations of their store components provide us with infinitely many possibilities for the selection of an initial network. For the examples in this section, we pick a set of three nodes $\mathcal{N}_0 = \{n_1, n_2, n_3\}$ and all four networks analysed have thus the following generic structure:

$$N^{(k)} \stackrel{\text{def}}{=} \|_{n \in \mathcal{N}_0}\ n[P^{(k)}(n), S^{(k)}]$$

$N^{(k)}$ denotes the network associated with protocol k. Nodes of these networks run the protocol $P^{(k)}(n) \stackrel{\text{def}}{=} Q_1^{(k)}(n) \mid Q_2^{(k)}(n)$ which consists of two parts: an active component $Q_1^{(k)}(n)$ which can initiate a routing message, and a reactive component $Q_2^{(k)}(n)$ which handles routing messages coming from other sources. $Q_1^{(k)}(n)$ and $Q_2^{(k)}(n)$ are specified below for $k = 1, \ldots, 4$. $S^{(k)}$ is the initialisation of the store component of a node in network k, where $S^{(k)} = \varepsilon$ for $k = 1, \ldots, 3$ and $S^{(4)}$ is specified below.

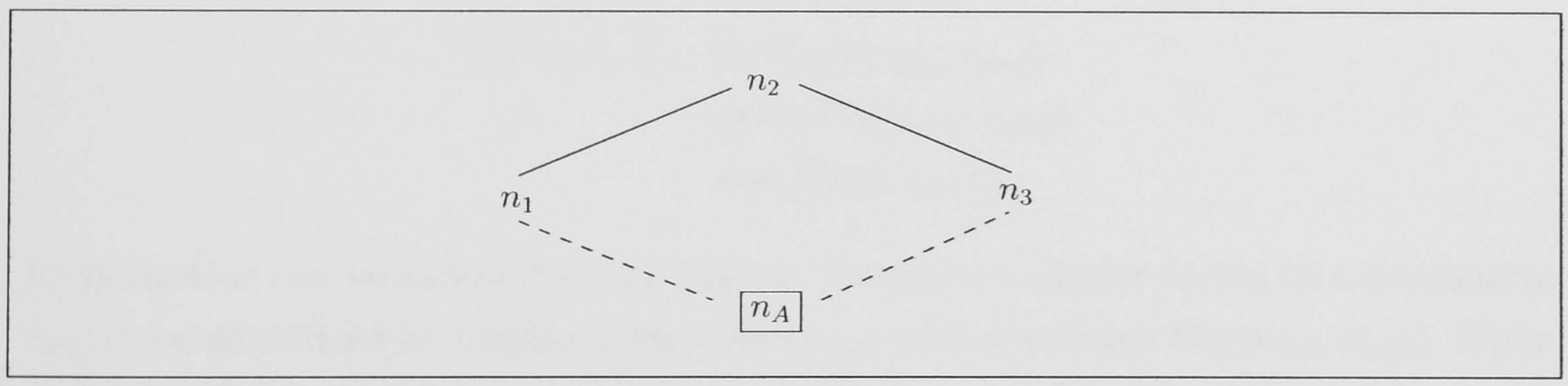

Figure 5.3: Protocols 1–4: network topology

We choose the three parameters of Definition 5.10 as follows:

(TC1) $A(1, n_A)$

(TC2) $\mathcal{T} = \{G_1, G_2, G_3\}$, where $V(G_i) = \mathcal{N}_0 \cup \{n_A\}$ for $i = 1, \ldots, 3$ and

$$\begin{aligned} E(G_1) &= \{\{n_1, n_2\}, \{n_2, n_3\}, \{n_1, n_A\}\} \\ E(G_2) &= \{\{n_1, n_2\}, \{n_2, n_3\}, \{n_A, n_3\}\} \\ E(G_3) &= \{\{n_1, n_2\}, \{n_2, n_3\}, \{n_1, n_A\}, \{n_A, n_3\}\} \end{aligned}$$

(TC3) As defined in Example 5.9

The parametric attacker $A(1, n_A)$ owns one node, knows the name n_A but no cryptographic information; its location is denoted n_A. $\mathcal{T}$ is depicted in Figure 5.3 where unstable links are drawn as dashed lines.

In the remainder of this section, we proceed for each of the four toy protocols according to the same scheme. We first model and explain the protocol process $P^{(k)}(n)$. Then the following analyses are computed:

$$(\kappa^\star, \sigma^\star, \gamma) \vDash_{\mathcal{G}(\mathcal{T})} N^{(k)} \parallel A(1, n_A)$$
$$(\kappa^\star, \sigma^\star, \gamma) \vDash_{\mathcal{G}(\mathcal{T})} N^{(k)}$$

We present the analyses in result tables showing selected components of the analysis estimates: $\sigma^\star$, where all elements are shown; and γ, where the relevant rules are shown, i.e. with respect to $\sigma^\star$. The label ℓ_{n_A} represents a generic label owned by the attacker. Shaded entries in the result tables represent valid estimates for both analyses, other entries are valid estimates for the first analysis (with attacker). Using the analysis results, we then apply Theorem 5.11 or Theorem 5.12 in order to establish topology consistency or inconsistency of the protocol. In case of topology inconsistency, we also describe a related attack.

Protocol 1. This first protocol makes no use of any cryptographic techniques to secure its routing behaviour, and it is therefore expected to be vulnerable to attacks.

$$\begin{aligned} Q_1^{(1)}(n) &\stackrel{\text{def}}{=} \textbf{!out}\ \mathsf{Msg}(n, n) \\ Q_2^{(1)}(n) &\stackrel{\text{def}}{=} \textbf{!in}\ \mathsf{Msg}(; x_{dst}, x_{snd}). \\ &\quad\ \textbf{store}\ \mathsf{Rte}(x_{dst}, x_{snd}). \\ &\quad\ \textbf{out}\ \mathsf{Msg}(x_{dst}, n) \end{aligned}$$

Its behaviour can be summarised as follows: A node n waits for routes to a destination n_{dst} to be advertised by neighbouring nodes n_{snd} with a message $\mathsf{Msg}(n_{dst}, n_{snd})$. Upon reception of this message, n makes an entry $\mathsf{Rte}(n_{dst}, n_{snd})$ to its routing table to note that n_{snd} is on a route to n_{dst}, and rebroadcasts $\mathsf{Msg}(n_{dst}, n)$ to announce that it knows a route to n_{dst}. A node n may also spontaneously advertise a route to itself by sending $\mathsf{Msg}(n, n)$.

Table 5.4 shows the estimate of the analysis of Protocol 1. As expected, network $N^{(1)}$ turns out not to be topology consistent.

Proposition 5.13 *Network $N^{(1)}$ is not topology consistent under (TC1)–(TC3).*

Proof. From the definition of $A(1, n_A)$ it is clear that it can emulate the following much

$\sigma^\star$	γ			
(ℓ_1, n_1)	$\ell_e \to \varepsilon$	$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_1})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_A})$
(ℓ_2, n_2)	$\ell_{n_A} \to \ldots$	$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_3})$
(ℓ_3, n_3)	$\ell_{n_1} \to n_1$	$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_A})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_1})$
(ℓ_{n_A}, n_A)	$\ell_{n_2} \to n_2$	$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_3})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_2})$
(ℓ_ε, n_1)	$\ell_{n_3} \to n_3$	$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_1})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_A})$
(ℓ_ε, n_2)		$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_3})$
(ℓ_ε, n_3)	$\ell_1 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_3})$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_A})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_1})$
	$\ell_1 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_1})$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_3})$		$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_2})$
	$\ell_1 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_2})$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_A})$
	$\ell_1 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_A})$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_2})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_3})$
	$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_A})$		$\ell_3 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_A})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_1})$
	$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_3})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_2})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_2})$

Table 5.4: Analysis estimate: Protocol 1

simpler attacker:

$$A \stackrel{\text{def}}{=} n_A[\textbf{in}\ \mathsf{Msg}(; x_1, x_2).\,\textbf{out}\ \mathsf{Msg}(x_1, n_A).\,\textbf{nil}, \varepsilon]$$

We have the following derivation (where $N_2^{(1)'}$, $N_3^{(1)'}$, etc. are abbreviations which can be easily substituted):

$$N^{(1)} \parallel A \xrightarrow{(\mathsf{Msg}(n_3,n_3),n_3)!}_{G_2 \in \mathcal{T}} N_1^{(1)} \parallel N_2^{(1)'} \parallel N_3^{(1)'} \parallel n_A[\textbf{out}\ \mathsf{Msg}(n_3, n_A).\,\textbf{nil}, \varepsilon]$$
$$\xrightarrow{(\mathsf{Msg}(n_3,n_A),n_A)!}_{G_1 \in \mathcal{T}} n_1[P^{(1)'}(n_1), \mathsf{Rte}(n_3, n_A)] \parallel N_2^{(1)''} \parallel N_3^{(1)'} \parallel n_A[\textbf{nil}, \varepsilon]$$

Thus with Conv-Comm we have $(N^{(1)} \parallel A(1, n_A)) \Downarrow_{n_1}^{\mathcal{T}} \mathsf{Rte}(n_3, n_A)$. On the other hand, $\mu_{n_1}^{\mathcal{G}(\mathcal{T})}(\mathsf{Rte}(n_3, n_A)) \neq \varepsilon$ and $\mathsf{Rte}(n_3, n_A) \notin L(\gamma, \ell)$ for $(\ell, n_1) \in \sigma^\star$ for the analysis of $N^{(1)}$ alone (shaded entries in Table 5.4). With Theorem 5.12, the result follows. □

In general, every breach of topology consistency entails a multitude of possible attacks which depend, among other conditions, on the respective network topology. The attack presented in the proof of Proposition 5.13 is based on the fact that both n_1 and n_3 believe to be connected by a path via a honest protocol participant n_A, which is not the case. However, it is clear that the attacker could have used arbitrary values instead of the name n_A. This suggests a presumably more powerful attack:

Attack 5.14 (Routing Loop) $A(1, n_A)$ can perform the following steps:

(1) Eavesdropping on the routing activity to discover that n_1 and n_2 are directly connected and that $A(1, n_A)$ can directly connect to n_1

(2) Impersonating n_2 and offering a route to a fictitious node n_0 whilst connected to n_1 with the process **out** $\mathsf{Msg}(n_0, n_2)$. **nil**

(3) Sending to n_1 a data packet addressed to n_0

The network responds in the following way:

1. After step (2), n_1 stores $\mathsf{Rte}(n_0, n_2)$ and sends $\mathsf{Msg}(n_0, n_1)$
2. On reception of $\mathsf{Msg}(n_0, n_1)$ from n_1, n_2 stores $\mathsf{Rte}(n_0, n_1)$ and sends $\mathsf{Msg}(n_0, n_2)$

At this point the network looks like in Figure 5.4, where the arrows labelled $\mathbf{n_0}$ and pointing to n_i symbolise routing table entries $\mathsf{Rte}(n_0, n_i)$.

3. On reception of the data packet, n_1 and n_2 continue forwarding this packet in a circle (routing loop attack, see §1.1.4). ⊠

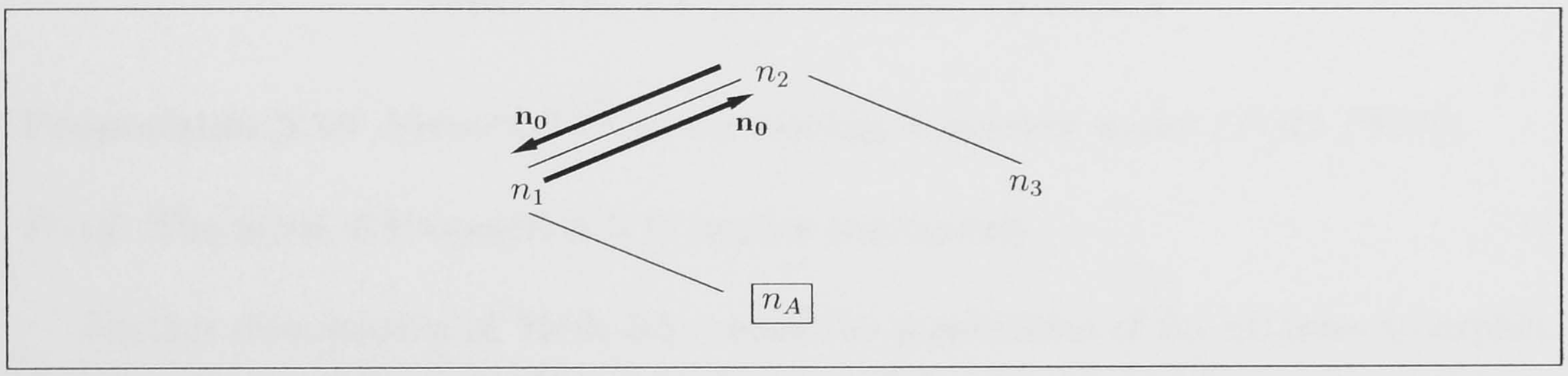

Figure 5.4: Protocol 1: routing loop attack

At this point, it is again important to emphasise that we use the toy protocols in this section for illustration only. Indeed, Protocol 1 is so simplistic that it might insert a routing loop even in a run without any attacker. This can be observed from the analysis results as well when considering the shaded entries only, in which case, for instance, one can find a loop between n_1 and n_2 for destination n_3.

Protocol 2. Trying to prevent Attack 5.14, encountered on Protocol 1, one might try to authenticate endpoint nodes with digital signatures. Protocol 2 is the result of this approach:

$$
\begin{aligned}
Q_1^{(2)}(n) &\stackrel{\text{def}}{=} \textbf{!out}\ \mathsf{Msg}(n, \mathsf{Sign}(\mathsf{PrivKey}(n), n), n) \\
Q_2^{(2)}(n) &\stackrel{\text{def}}{=} \textbf{!in}\ \mathsf{Msg}(; x_{dst}, x_{sig}, x_{snd}). \\
&\qquad \textbf{checksig}\ x_{sig}\ \mathsf{PubKey}(x_{dst})\ [x_{dst}] \\
&\qquad\qquad \textbf{store}\ \mathsf{Rte}(x_{dst}, x_{snd}). \\
&\qquad\qquad \textbf{out}\ \mathsf{Msg}(x_{dst}, x_{sig}, n) \\
&\qquad \textbf{else nil}
\end{aligned}
$$

The existence of a public key infrastructure is modelled as in §3.4.2. Otherwise Protocol 2 resembles Protocol 1 with the exception that messages now contain the signature of the destination node as additional component. Before storing a routing table entry and rebroadcasting an incoming message, nodes verify first that the signature is valid and abort the procedure if it is found invalid.

Considering the analysis estimate displayed in Table 5.5, it turns out once again that $N^{(2)}$ is not topology consistent.

$\sigma^\star$	γ			
(ℓ_1, n_1)	$\ell_e \to \varepsilon$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_3})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_3})$
(ℓ_2, n_2)	$\ell_{n_A} \to \ldots$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_1})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_1})$
(ℓ_3, n_3)	$\ell_{n_1} \to n_1$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_A})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_A})$
(ℓ_{n_A}, n_A)	$\ell_{n_2} \to n_2$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_2})$
(ℓ_ε, n_1)	$\ell_{n_3} \to n_3$	$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_A})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_A})$
(ℓ_ε, n_2)		$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_3})$		$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_3})$
(ℓ_ε, n_3)	$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_A})$	$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_A})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_1})$
	$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_3})$	$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_2})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_2})$
	$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_1})$		$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_1})$	
	$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_2})$	

Table 5.5: Analysis estimate: Protocol 2

Proposition 5.15 *Network $N^{(2)}$ is not topology consistent under (TC1)–(TC3).*

Proof. The proof of Proposition 5.13 applies analogously. □

Further examination of Table 5.5 reveals the possibilities of the attacker to exploit the flaw. In comparison to Table 5.4, the sizes of the sets $L(\gamma, \ell_i)$ for $i = 1, \ldots, 3$ have decreased: for instance, it is no longer possible for the attacker to insert tuples with arbitrary destination, which are symbolised in Table 5.4 by the possibility of having ℓ_{n_A} as destination. This is because the attacker is not able to produce new signatures for fictitious destinations. However, the attacker can reuse the given signatures in order to talk about a valid destination, but forge the address of the immediate sender. Thus $A(1, n_A)$ can make route table entries for valid destinations point to any destination. While this means that Attack 5.14 is no longer possible in this scenario, the attacker may perform the following simpler attack:

Attack 5.16 (Grey Hole) $A(1, n_A)$ can perform the following steps:

(1) Eavesdropping on the routing activity to discover that n_1 and n_3 are not directly connected and that $A(1, n_A)$ can directly connect to both of them

(2) Eavesdropping to acquire $\mathsf{Sign}(\mathsf{PrivKey}(n_1), n_1)$ and $\mathsf{Sign}(\mathsf{PrivKey}(n_3), n_3)$

(3) Impersonating n_3 and offering a route to n_3 whilst connected to n_1 with the process **out** $\mathsf{Msg}(n_3, \mathsf{Sign}(\mathsf{PrivKey}(n_3), n_3), n_3)$. **nil**

(4) Impersonating n_1 and offering a route to n_1 whilst connected to n_3 with the process **out** $\mathsf{Msg}(n_1, \mathsf{Sign}(\mathsf{PrivKey}(n_1), n_1), n_1)$. **nil**

The network responds in the following way:

1. n_1 stores $\mathsf{Rte}(n_3, n_3)$ and sends $\mathsf{Msg}(n_3, n_1)$

2. n_3 stores $\mathsf{Rte}(n_1, n_1)$ and sends $\mathsf{Msg}(n_1, n_3)$

At this point the network looks like in Figure 5.5. Nodes n_1 and n_3 might prefer this route over possible other discovered routes because they believe that they are directly connected to each other. Whenever one of them decides to send a data message to the other, the attacker may impersonate the other node, acknowledge receipt, but not forward the message. Thus, all sent messages between the two participants are lost and the loss is not detected (grey hole attack, see §1.1.4). Of course, this attack might no longer be possible if the transport layer is secured as well. ⊠

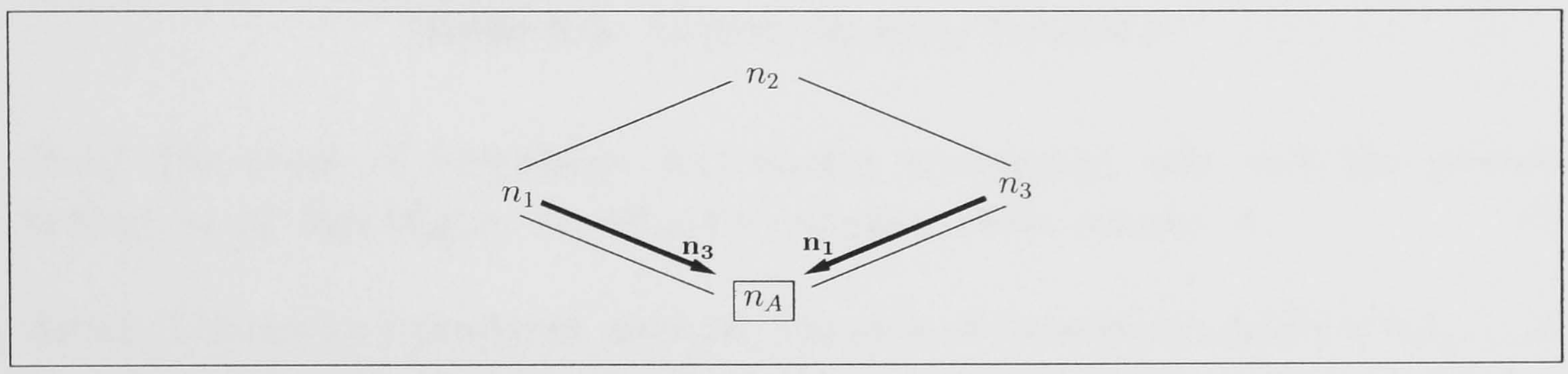

Figure 5.5: Protocol 2: grey hole attack

Protocol 3. This protocol tries to overcome the deficiencies of Protocol 2 by adding a second signature to authenticate the next hop information.

$$
\begin{aligned}
Q_1^{(3)}(n) &\stackrel{\text{def}}{=} \textbf{!out}\ \mathsf{Msg}(n, \mathsf{Sign}(\mathsf{PrivKey}(n), n), n, \\
&\qquad \mathsf{Sign}(\mathsf{PrivKey}(n), n, \mathsf{Sign}(\mathsf{PrivKey}(n), n), n)) \\
Q_2^{(3)}(n) &\stackrel{\text{def}}{=} \textbf{!in}\ \mathsf{Msg}(; x_{dst}, x_{sig_1}, x_{snd}, x_{sig_2}). \\
&\quad \textbf{checksig}\ x_{sig_1}\ \mathsf{PubKey}(x_{dst})\ [x_{dst}] \\
&\quad\quad \textbf{checksig}\ x_{sig_2}\ \mathsf{PubKey}(x_{snd})\ [x_{dst}, x_{sig_1}, x_{snd}] \\
&\quad\quad\quad \textbf{store}\ \mathsf{Rte}(x_{dst}, x_{snd}). \\
&\quad\quad\quad \textbf{out}\ \mathsf{Msg}(x_{dst}, x_{sig_1}, n, \mathsf{Sign}(\mathsf{PrivKey}(n), x_{dst}, x_{sig_1}, n)) \\
&\quad\quad \textbf{else nil} \\
&\quad \textbf{else nil}
\end{aligned}
$$

Protocol 3 is thus derived from Protocol 2 by adding a signature of the tuple of the complete message (destination, destination signature, sender) computed with the private key of the sender to the message. Upon reception and verification of the destination signature, the message signature is checked as well, using the public key of the sender. Only if both checks succeed, the node updates its routing table and rebroadcasts the routing message.

It turns out that even adding two layers of signatures in this way is not enough to make the original Protocol 1 topology consistent:

Proposition 5.17 *Network $N^{(3)}$ is not topology consistent under (TC1)–(TC3).*

$\sigma^\star$	γ			
(ℓ_1, n_1)	$\ell_e \to \varepsilon$	$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_1})$
(ℓ_2, n_2)	$\ell_{n_A} \to \ldots$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_3})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_2})$
(ℓ_3, n_3)	$\ell_{n_1} \to n_1$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_1})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_3})$
(ℓ_{n_A}, n_A)	$\ell_{n_2} \to n_2$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_1})$
(ℓ_ε, n_1)	$\ell_{n_3} \to n_3$	$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_3})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_2})$
(ℓ_ε, n_2)		$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_1})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_3})$
(ℓ_ε, n_3)	$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_3})$	$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_2})$		$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_1})$
	$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_1})$		$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_2})$

Table 5.6: Analysis estimate: Protocol 3

Proof. The proof of Proposition 5.13 applies analogously, take now the process **in** $\mathsf{Msg}(; x_1, x_2)$. **out** $\mathsf{Msg}(x_1, x_2)$. **nil** in the definition of the attacker A. □

Attack 5.16 can be reproduced, although the sizes of the sets $L(\gamma, \ell_i)$ for $i = 1, \ldots, 3$ have decreased again in comparison to Table 5.5: now neither destination nor sender name can be set to invalid values by n_A. However, this does not prevent the attacker from replaying messages and the attacker can make route table entries for valid destinations point to any valid destination.

Protocol 4. This protocol is almost identical to Protocol 3, however honest nodes have now the ability to know with certainty who their valid neighbours are. In practice, this could be achieved for example by a secure Layer 2 protocol or, in the non-mobile case, if nodes are hardwired.

$$N^{(4)} \stackrel{\text{def}}{=} n_1[P^{(4)}(n_1), \mathsf{Nb}(n_2)] \parallel n_2[P^{(4)}(n_2), \mathsf{Nb}(n_1) \cup \mathsf{Nb}(n_3)] \parallel n_3[P^{(4)}(n_3), \mathsf{Nb}(n_2)]$$

In our modelling, this assumption is expressed by store items $\mathsf{Nb}(n_i)$ telling a node that it has neighbour n_i.

$$\begin{aligned}
Q_1^{(4)}(n) &\stackrel{\text{def}}{=} Q_1^{(3)}(n) \\
Q_1^{(4)}(n) &\stackrel{\text{def}}{=} \textbf{!in}\ \mathsf{Msg}(; x_{dst}, x_{sig_1}, x_{snd}, x_{sig_2}). \\
&\quad \textbf{checksig}\ x_{sig_1}\ \mathsf{PubKey}(x_{dst})\ [x_{dst}] \\
&\quad\quad \textbf{checksig}\ x_{sig_2}\ \mathsf{PubKey}(x_{snd})\ [x_{dst}, x_{sig_1}, x_{snd}] \\
&\quad\quad\quad \textbf{read}\ \mathsf{Nb}(x_{snd};) \\
&\quad\quad\quad\quad \textbf{store}\ \mathsf{Rte}(x_{dst}, x_{snd}). \\
&\quad\quad\quad\quad \textbf{out}\ \mathsf{Msg}(x_{dst}, x_{sig_1}, n, \mathsf{Sign}(\mathsf{PrivKey}(n), x_{dst}, x_{sig_1}, n)) \\
&\quad\quad\quad \textbf{else nil} \\
&\quad\quad \textbf{else nil} \\
&\quad \textbf{else nil}
\end{aligned}$$

The routing process $Q_1^{(4)}(n)$ is identical to $Q_1^{(3)}(n)$, $Q_1^{(4)}(n)$ contains the additional check whether the node claiming to be the sender of a routing message is indeed one of the neighbours of the current node.

$\sigma^\star$		γ			
(ℓ_1, n_1)	(ℓ_6, n_2)	$\ell_e \to \varepsilon$	$\ell_4 \to \mathsf{Nb}(n_1)$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_1})$
(ℓ_2, n_2)	(ℓ_5, n_3)	$\ell_{n_A} \to \ldots$	$\ell_5 \to \mathsf{Nb}(n_2)$		$\ell_2 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_3})$
(ℓ_3, n_3)	(ℓ_ε, n_1)	$\ell_{n_1} \to n_1$	$\ell_6 \to \mathsf{Nb}(n_3)$	$\ell_2 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_3})$	
(ℓ_{n_A}, n_A)	(ℓ_ε, n_2)	$\ell_{n_2} \to n_2$		$\ell_2 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_2})$
(ℓ_5, n_1)	(ℓ_ε, n_3)	$\ell_{n_3} \to n_3$	$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_1})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_2})$
(ℓ_4, n_2)			$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_3})$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_2})$

Table 5.7: Analysis estimate: Protocol 4

The analysis estimate is to be found in Table 5.7. The analyses of $N^{(4)}$ alone and with attacker almost coincide which gives an indication that the protocol might be topology consistent. We can prove this fact as follows:

Proposition 5.18 *Network $N^{(4)}$ is topology consistent under (TC1)–(TC3).*

Proof. According to Theorem 5.11, we have to check for $n \in V(N)$ and $(\ell, n) \in \sigma^\star$ that for all $U \in L(\gamma, \ell)$ the equation $\mu_n^{\mathcal{G}(\mathcal{T})}(U) = \varepsilon$ is true. This is easily established by inspection of the (finite) sets $L(\gamma, \ell)$. □

Note that Table 5.7 has been computed using the (untyped) original analysis, according to our policy regarding the use of the typed analysis (see §5.1.3); the typed analysis yields however the same set of estimates which are relevant for the security evaluation.

5.3.2 Alternative Attack Scenarios

The four protocols of §5.3.1 have been analysed using a single adversarial scenario ($A(1, n_A)$ attacker), and Protocols 1–3 have been found to be flawed whereas Protocol 4 proved to be topology consistent. Following our discussion in §5.1.2, it might therefore be interesting to evaluate Protocols 1–3 using weaker attacker models and Protocol 4 using stronger ones. In order to remain comparable with the previous section we retain conditions (TC2) and (TC3) but change condition (TC1) accordingly.

Passive Attacker. A passive attacker only eavesdrops on communication without sending its own messages. While eavesdropping is an important part of the successful attacks we have found so far, topology consistency can obviously not be influenced in this manner. It is however interesting to note that our Control Flow Analysis is precise enough to establish topology consistency for Protocols 1–3 in presence of a passive attacker and conditions (TC2) and (TC3), which becomes evident when examining the shaded entries of Tables 5.4–5.6.

Attacker *Active*-1-1. We formally exchange condition (TC1) to the following:

(TC1′) $A(1, \mathsf{PrivKey}(n_2))$

Thus the attacker got hold of the private key of the node n_2. Interestingly, although the attacker possesses now a valid key of the system, the system is still resilient against it: the analysis result is identical to Table 5.7.

Proposition 5.19 *Network $N^{(4)}$ is topology consistent under (TC1′), (TC2), (TC3).*

Proof. The proof of Proposition 5.18 applies analogously. □

The reason for this is that the attacker, although being able to forge both kinds of signatures using $\mathsf{PrivKey}(n_2)$ can only be successful with signatures of the form $\mathsf{Sign}(\mathsf{PrivKey}(n'), n')$ and $\mathsf{Sign}(\mathsf{PrivKey}(n_2), \mathsf{Tup}(n', \mathsf{Sign}(\mathsf{PrivKey}(n'), n'), n_2)))$ in the same message, where n' is arbitrary. As the attacker cannot produce destination signatures other than the ones which can be discovered by eavesdropping, *Active*-1-1 is not more powerful than $A(1, n_A)$. Note that the results do not change if *Active*-1-1 is chosen to be $N^{(A)}(1, \mathsf{PrivKey}(n_0))$, meaning that the attacker is in possession of a private key of a node not present in the network.

Attacker *Active*-2-1. Parameter (TC1) is now changed to

(TC1″) $A(1, \mathsf{PrivKey}(n_2) \cup \mathsf{PrivKey}(n_0))$

thus assuming that the attacker possesses the private key of the node n_2 and of a node n_0, which is accepted as a honest node in the system, but is not present in the system at the moment.

$\sigma^\star$		γ			
(ℓ_1, n_1)	(ℓ_6, n_2)	$\ell_e \to \varepsilon$	$\ell_4 \to \mathsf{Nb}(n_1)$	$\ell_3 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_1})$
(ℓ_2, n_2)	(ℓ_5, n_3)	$\ell_{n_A} \to n_0$	$\ell_5 \to \mathsf{Nb}(n_2)$	$\ell_1 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_1})$
(ℓ_3, n_3)	(ℓ_ε, n_1)	$\ell_{n_A} \to \ldots$	$\ell_6 \to \mathsf{Nb}(n_3)$	$\ell_1 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_3})$
(ℓ_{n_A}, n_A)	(ℓ_ε, n_2)			$\ell_1 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_1})$
(ℓ_5, n_1)	(ℓ_ε, n_3)	$\ell_{n_1} \to n_1$	$\ell_3 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_2})$	$\ell_1 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_2}, \ell_{n_3})$
(ℓ_4, n_2)		$\ell_{n_2} \to n_2$	$\ell_3 \to \mathsf{Rte}(\ell_{n_A}, \ell_{n_2})$		$\ell_2 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_1})$
		$\ell_{n_3} \to n_3$	$\ell_3 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_2})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_1}, \ell_{n_3})$	$\ell_2 \to \mathsf{Rte}(\ell_{n_3}, \ell_{n_3})$

Table 5.8: Analysis estimate: variant scenario for Protocol 4

Being in the possession of two valid keys finally allows the attacker to break topology consistency of Protocol 4, and a routing loop attack becomes possible:

Attack 5.20 (Routing Loop) $A(1, \mathsf{PrivKey}(n_2) \cup \mathsf{PrivKey}(n_0))$ can perform the following steps:

(1) Eavesdropping on the routing activity to find out that n_1 and n_2 are directly connected and that n_A can directly connect to n_1

(2) Computing signatures $sig_1 = \mathsf{Sign}(\mathsf{PrivKey}(n_0), n_0)$ and $sig_2 = \mathsf{Sign}(\mathsf{PrivKey}(n_2), \mathsf{Tup}(n_0, sig_1, n_2), n_2))$

(3) Impersonating n_2 and offering a route to a fictitious node n_0 whilst connected to n_1 with a process **out** $\mathsf{Msg}(n_0, sig_1, n_2, sig_2)$. **nil**

(4) Sending to n_1 a data packet addressed to n_0

The network responds as stated before in Attack 5.14 and ends up with the routing loop of Figure 5.4. ⊠

Proposition 5.21 *Network $N^{(4)}$ is not topology consistent under (TC1″), (TC2), (TC3).*

Proof. The structure of the proof of Proposition 5.13 applies analogously. The attacker has to be constructed as outlined in Attack 5.20. □

Chapter 6

Case Studies

Communication protocols define a sequence of interactions between network entities to provide network services of some sort. These services can be quite complex and so are the resulting protocols with respect to both exchanged data and internal computation, making protocol design and implementation a challenge. In the networking community, the use of abstraction hierarchies for protocol development as well as detailed specifications, which are for instance published as Requests for Comments (RFCs) by the Internet Engineering Task Force (IETF), have long become tools for conquering this complexity. However, the level of detail varies within such specification documents: while data formats are specified to the last bit – a necessity to ensure interoperability of competing implementations – the data processing and the interactions are usually described in prose. This loss of formality might have to be paid for with flawed systems, especially when competing implementations interpret omissions in different ways.

In this chapter we report on our experiences with fully formal protocol specifications by applying our framework to two of the more prominent secure routing protocols for mobile ad-hoc networks. We have chosen SAODV and ARAN (see §2.3.2) which are both based on an AODV-like routing technique and thus make for a particularly good comparison. In order to keep the analysis tractable (see §4.5.5), we decompose the protocols into parts and carefully select critical message components. The modelling of the protocol parts, the analysis results, and conclusions for the security of the protocol are described in the chapter's two sections §6.1 (SAODV) and §6.2 (ARAN). The complete models of all protocol parts discussed in this chapter can be found in Appendix B.

6.1 SAODV

The *Secure Ad-Hoc On-Demand Distance Vector (SAODV)* routing protocol [61, 59] is a secure extension of AODV [117, 115], which we have described before in §1.1.3 as one of the leading approaches to routing in mobile ad-hoc networks. For each of the message types of AODV, the extension defines additional message fields which contain

signatures, hash values, and public keys and prescribes the processing of these values. SAODV is mainly based on two mechanisms [61]: *digital signatures* "to protect the non-mutable fields of routing messages", and *hash chains* "to secure the hop count information", which is mutable. We analyse the two mechanisms separately in §6.1.1 and §6.1.2, using two specifications which we call μSAODV_RD_DS and μSAODV_RD_HC, respectively. Both these models are concerned with route discovery only. In §6.1.3 we complement this with μSAODV_RM, a model of the route maintenance part of SAODV.

6.1.1 Route Discovery: Digital Signatures

Distance vector routing protocols such as AODV are characterised by describing topological information by routing table entries which store direction (next hop) and distance (hop count) for every destination. In this section we describe and analyse a model μSAODV_RD_DS in order to investigate whether SAODV's digital signatures provide an effective way of securing direction information for AODV during route discovery.

Operation and Modelling. During a route discovery step a node *src* (the *source*) tries to reach a node *dst* (the *destination*) using a *route request (RREQ)* message, whereupon *dst* answers with a *route reply (RREP)* message; while these messages traverse the network, reverse and forward routes between *src* and *dst* are established.

In the following we present the source code of μSAODV_RD_DS (as it is used for input to our prototype; see also §B.1.1) along with additional explanations. Function declarations are defined first:

```
1 signature RREQ(dst : τn, src : τn, snd : τn, sig1 : τsign1) : τrreq message
2 signature RREP(dst : τn, src : τn, addr : τn, snd : τn, sig2 : τsign2) : τrrep message
```

Lines 1 and 2 show the format of the involved routing messages, where the message components denote the following:

dst	IP address of the destination of the RREQ
src	IP address of the source of the RREQ
snd	IP address of the sender of the message (part of IP header)
addr	IP address of the addressee of the message (part of IP header)
sig_1	$\mathsf{Sign}_1(\mathsf{PrivKey}(src), dst, src)$
sig_2	$\mathsf{Sign}_2(\mathsf{PrivKey}(dst), dst, src)$

All IP addresses are thus uniformly modelled by names (of type τ_n). Note that the highlighted address components are not mentioned in the AODV or SAODV header specifications as they are taken instead from the IP header, which serves as an enve-

lope for the routing messages. These components are nonetheless indispensable for the acquisition of direction information and are therefore included in the modelling. The components *snd* and *addr* represent the hop-to-hop addressing of the message: for example, the component *addr* of RREP implements a unicast, whereas in the RREQ this component is omitted to mean that all messages are implicitly broadcast (i.e., *addr* is always set to the IP limited broadcast address `255.255.255.255`). So far, all discussed components are part of the AODV and IP specification. SAODV's contribution in this model consists of the signatures sig_1 and sig_2 of all AODV components (but not the IP header!). The signatures can be uniquely associated with either RREQ or RREP, which is realised in SAODV by signing the type of the message, but having distinct functions Sign_1 and Sign_2 does the trick in our modelling.

```
3  signature PubKey(τ_n) : τ_pubkey secret
4  signature PrivKey(τ_n) : τ_privkey secret
5  signature Sign1(τ_privkey, τ_n, τ_n) : τ_sign1 oneway
6  signature Sign2(τ_privkey, τ_n, τ_n) : τ_sign2 oneway
7  signature Rte(τ_n, τ_n) : τ_rte secret
8  signature Keys(τ_n, τ_pubkey) : τ_rte secret
```

Lines 3–8 show the remaining function declarations, most of which are straightforward. Rte and Keys are only used for internal storage and can therefore be kept secret in the modelling. $\mathsf{Rte}(dst, next)$ describes, as before, a routing table entry with destination *dst* and next hop *next*, $\mathsf{Keys}(n, pubkey)$ holds the public key *pubkey* for every node n.

We describe the operation of μSAODV_RD_DS from the point of view of an arbitrary node n, which can have any of three roles: source, destination, or intermediate node.

```
10 process SendRREQ(n, dst) ≝
11     out RREQ(dst, n, n, Sign1(PrivKey(n), dst, n))
```

A node n initiates a route discovery by broadcasting a RREQ of correct format (line 11). Note that at this step n represents source *src* and immediate sender *snd* at the same time.

```
13 process ReceiveRREQ(n) ≝
14     in RREQ(; dst, src, snd, sig1).
15     read Keys(src; pubkey)
16         checksig1 sig1 pubkey [dst, src]
17             store Rte(src, snd).
18             if dst = n then
19                 out RREP(n, src, snd, n, Sign2(PrivKey(n), n, src))
20             else
21                 out RREQ(dst, src, n, sig1)
22         else nil
```

```
23  else nil
```

Every node n receiving a RREQ verifies that the alleged source *src* indeed signed the message with its private key (lines 15–16); n aborts if this check fails. Otherwise, n makes an entry into its routing table to provide a *reverse route* leading to *src* (l. 17). Node n then checks whether it is the destination of the RREQ, i.e. $dst = n$. If yes, n sends a RREP, signed with n's private key and addressed to the sender *snd* of the RREQ (l. 19). If no, n rebroadcasts the RREQ, changing only the address of the sender to its own address (l. 21).

```
25 process ReceiveRREP(n) ≝
26     in RREP(; dst, src, addr, snd, sig2).
27     if addr = n then
28         read Keys(dst; pubkey)
29             checksig2 sig2 pubkey [dst, src]
30                 store Rte(dst, snd).
31                 if src = n then
32                     nil
33                 else
34                     read Rte(src; next)
35                         out RREP(dst, src, next, n, sig2)
36                     else nil
37             else nil
38         else nil
39     else nil
```

Every node n receiving a RREP checks whether it is the addressee, i.e. $addr = n$; n aborts if this check fails (l. 39). Otherwise, n verifies that the alleged destination *dst* indeed signed the message with its private key (ll. 28–29); n aborts if this check fails. Otherwise, n makes an entry into its routing table to provide a *forward route* leading to *dst* (l. 30). If n was the source of the RREQ in the first place, i.e. $src = n$, it terminates because the route is established (l. 32). (n can start sending data packets to *dst* in this case.) Otherwise, n retrieves the next hop *next* on the reverse route to *src* from its routing table and resends the RREP addressed to *next*, changing only the address of the sender to its own address (ll. 34–35).

Analysis. A suitable scenario has to be defined before we can begin with the analysis. We choose the following:

```
41 process P(n, dst1, dst2) ≝
42     SendRREQ(n, dst1) | SendRREQ(n, dst2) | ReceiveRREQ(n) | ReceiveRREP(n)
43
44 network N(n, dst1, dst2) ≝
```

```
45    n[P(n, dst_1, dst_2), Keys(n, PubKey(n)) ∪ Keys(dst_1, PubKey(dst_1))∪
46          Keys(dst_2, PubKey(dst_2))]
47
48 network μSAODV_RD_DS ≝ N(n_1, n_2, n_3) ‖ N(n_2, n_1, n_3) ‖ N(n_3, n_1, n_2)
```

The scenario for μSAODV_RD_DS thus consists of three nodes n_1, n_2, and n_3 which each have the ability to send RREQs to their fellow nodes and receive and process RREQs and RREPs (ll. 41–42, 44–46). Furthermore, they can retrieve the public key of any node in the system (ll. 45–46).

```
50 topology
51    {n_1, n_2}
52    {n_2, n_3}
53    {n_1, ATT1}
54    {n_3, ATT1}
```

Lines 50–54 describe the abstract connectivity graph $\widehat{G}$ used by the analysis (see Notation 4.2); it is depicted in Figure 6.1. Note that *ATT1* is the generic name for a single attacker used by our implementation.

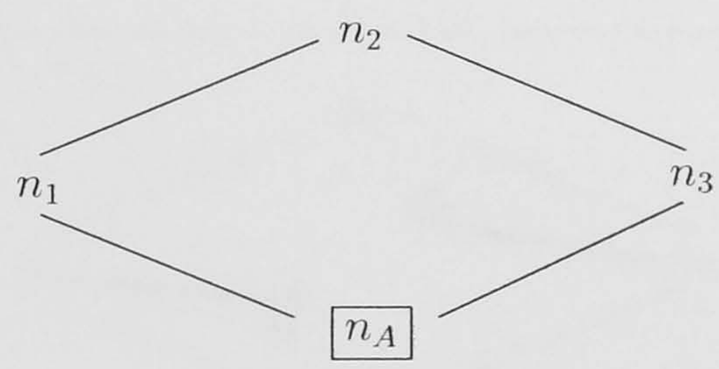

Figure 6.1: μSAODV_RD_DS: abstract connectivity graph

The consistency mediator of Example 5.9 can be easily adapted to the fact that we specify an abstract connectivity graph $\widehat{G}$ instead of a network topology.

```
56 attacker 1 [n_A]
```

The attacker is specified in line 56 as a parametric attacker $A(1, n_A)$ (see Notation 5.3) knowing the name n_A, which represents an arbitrary IP address.

Analogously to §5.3, we can compute the appropriate finite analyses with our prototype which are connected via Theorem 4.21 with the following analyses:

$$(\kappa^{(1)}, \sigma^{(1)}) \vDash_{\widehat{G}} \mu\text{SAODV_RD_DS}$$
$$(\kappa^{(2)}, \sigma^{(2)}) \vDash_{\widehat{G}} \mu\text{SAODV_RD_DS} \parallel A(1, n_A)$$

We omit giving the results of the finite analysis explicitly, but present the significant

analysis results directly (using Notation 4.4):

$$
\begin{aligned}
\sigma_{n_1}^{(1)} &= \{\mathsf{Rte}(n, n_2) \ : \ n \in \{n_1, n_2, n_3\}\} \\
\sigma_{n_2}^{(1)} &= \{\mathsf{Rte}(n, n') \ : \ n \in \{n_1, n_2, n_3\} \wedge n' \in \{n_1, n_3\}\} \\
\sigma_{n_3}^{(1)} &= \sigma_{n_1}^{(1)} \\
\sigma_{n_1}^{(2)} &= \{\mathsf{Rte}(n, n') \ : \ n \in \{n_1, n_2, n_3\} \wedge n' \in \{n_1, n_2, n_3, n_A\}\} \\
\sigma_{n_2}^{(2)} &= \sigma_{n_2}^{(1)} \\
\sigma_{n_3}^{(2)} &= \sigma_{n_1}^{(2)}
\end{aligned}
$$

The results indicate that network μSAODV_RD_DS might not be topology consistent, because both the stores of n_1 and n_3 may hold routing table entries pointing to the arbitrary destination n_A. This can be confirmed as follows:

Attack 6.1 The attacker can forward a RREQ from n_1 to n_3 unchanged except for the *snd* field which can be set to an arbitrary value (represented by n_A, which stands for any name the attacker might want to use), and same for the corresponding RREP. Nodes n_1 and n_3 believe that they have acquired a valid route via n_A, which does not hold for $n_A \neq n_2$. This situation is depicted in Figure 6.2, where the dangling pointers symbolise that routing table entries may point in undefined directions. ⊠

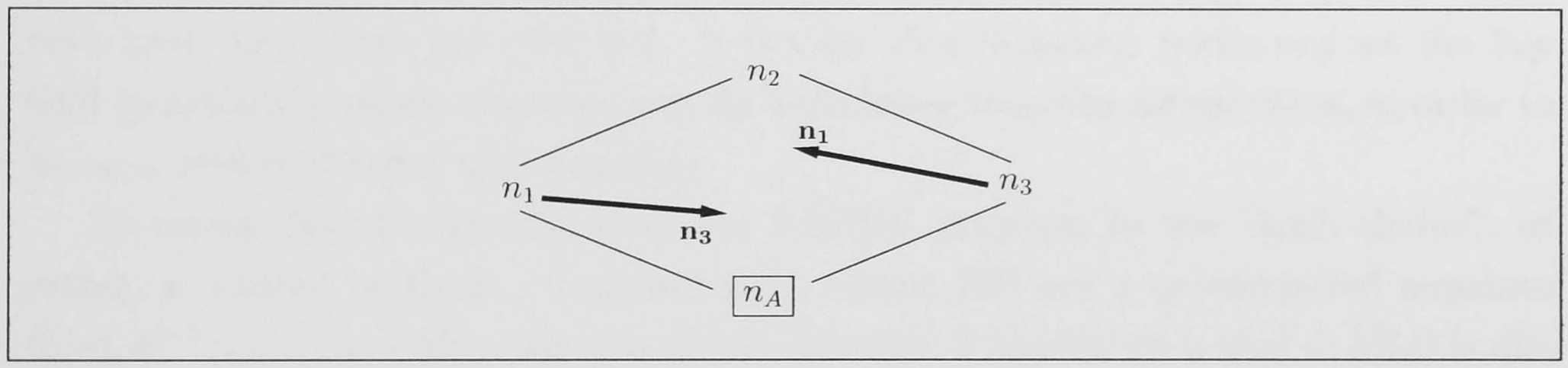

Figure 6.2: μSAODV_RD_DS: topology inconsistency

Conclusions. The question remains whether the attack encountered on μSAODV_RD_DS can be transferred to full SAODV. This seems to be possible since the attacker of Attack 6.1 only needs to forward messages unchanged except for one field. Additional components such as sequence numbers, request IDs, flags (which have been omitted to keep μSAODV_RD_DS small) do not have any influence on this fact.

It also seems that this flaw has first been detected by Papadimitratos and Haas who have suggested in [112] that the "IP portion of the S-AODV traffic can be trivially compromised", although they fail to point out the details and possible consequences. Guerrero Zapata and Asokan address this criticism in [61] as follows:

> "This [criticism by Papadimitratos and Haas] is not true because SAODV does not trust the contents of the IP header, and all the informa-

tion that [it] needs to operate is inside the AODV message and the SAODV extension."

The second part of this statement seems to be untrue, because the information about the next hop is *not* part of either AODV or SAODV headers and is taken from the IP header in implementations of AODV. We have subsequently reported these findings on the IETF MANET mailing list: Guerrero Zapata has then admitted that Attack 6.1 is possible, but considered it "out of scope" for SAODV since only limited damage is done. We believe this story illustrates nicely that formal models can help clarify the functionality of a protocol and are necessary for arguing about its correctness.

6.1.2 Route Discovery: Hash Chains

We describe and analyse a model μSAODV_RD_HC which complements the discussion of §6.1.1 by focusing on SAODV's means of securing distance information. Recall that this information is obtained in AODV as follows (compare §1.1.3): a separate message component for RREQ or RREP is dedicated to the hop count (*hop*), which describes the number of hops between the current receiver of the message and the originator; the originator sets the *hop* field to zero and intermediate nodes increment this field before sending the message on; the hop count is stored in routing tables and routes with lower hop count are preferred. It is clear that malicious nodes can set the *hop* field to arbitrary values, especially to the ones lower than the actual value, in order to be on a preferred route with certainty.

To secure this distance information, SAODV proposes to use "hash chains", or rather, a variant of them. Standard hash chains [85] are a precomputed sequence $h^i(s), h^{i-1}(s), \ldots, s$ with a one-way (hash) function h applied to a seed s: $h^i(s)$ is distributed to the entities that intend to authenticate the sender, and the sender can successively include values $h^{i-1}(s), \ldots, s$ in messages as tokens of authentication; receivers can authenticate the sender by hashing the received token the appropriate number of times to yield $h^i(s)$. In SAODV this idea is modified by having the seed published first and successively hashed as it traverses the network. This method is proposed "to verify that the hop count has not been decremented by an attacker" [61].

Operation and Modelling. μSAODV_RD_HC models the route request step of AODV/SAODV only.

```
1 signature RREQ(src : τn, hop : τint, maxhop : τint, tophash : τhash, sig : τsign,
                 hash : τhash) : τrreq message
```

The type declaration shows that we have omitted direction information completely in order to keep μSAODV_RD_HC small. The message components have the following

meanings, where all addresses are modelled again uniformly by names:

src	IP address of the source of the RREQ
hop	Hop count
maxhop	Maximum hop count
tophash	$\mathsf{Hash}(maxhop, \mathsf{Seed}(src))$
sig	$\mathsf{Sign}(\mathsf{PrivKey}(n), src, maxhop, tophash)$
hash	$\mathsf{Hash}(hop, \mathsf{Seed}(src))$

The components *hop* and *maxhop* are introduced to describe the current and maximum hop count. The component *maxhop* is a fixed parameter of AODV (the assumed maximum diameter of a network) and is set to 2 in our case. Both *hop* and *maxhop* are integer values of type τ_{int} and modelled as usual:

```
2 signature Zero() : τint twoway
3 signature Succ(τint) : τint twoway
```

For readability, we introduce the following translation into numerals:

$$\begin{aligned} \#0 &\stackrel{\text{def}}{=} \mathsf{Zero}() \\ \#\mathsf{i} &\stackrel{\text{def}}{=} \mathsf{Succ}(\#(\mathsf{i}-1)) \end{aligned}$$

The components *tophash* and *hash* on the other hand are hash values of type τ_{hash}:

```
4 signature Seed(τn) : τseed secret
5 signature Hash(τint, τseed) : τhash oneway
```

$\mathsf{Hash}(exp, seed)$ describes the application of the hash function to *seed* for *exp*-times. The seed is modelled via a secret function Seed which can be used to attribute a seed to every node in the network.

```
8 signature Sign(τprivkey, τn, τint, τhash) : τsign oneway
9 signature Rte(τn, τint) : τrte secret
```

The signature *sig* signs all but the mutable (*hop*, *hash*) components of the message. Note the change in the format of the routing table entry $\mathsf{Rte}(dst, hop)$ which records now the hop count for every destination.

```
12 process SendRREQ(n, dst) ≝
13     out RREQ(n, #0, #2, Hash(#2, Seed(n)),
14         Sign(PrivKey(n), n, #2, Hash(#2, Seed(n))), Hash(#0, Seed(n)))
```

Lines 13–14 express that a node n initiates a route discovery by broadcasting a RREQ

of correct format.

```
16 process ReceiveRREQ(n) ≝
17     in RREQ(; src, hop, maxhop, tophash, sig, hash).
18     read Keys(src; pubkey)
19         checksig sig pubkey [src, maxhop, tophash]
20             case tophash of Hash(maxhop; seed)
21                 case hash of Hash(hop, seed; )
22                     store Rte(src, hop).
23                     out RREQ(src, Succ(hop), maxhop, tophash, sig,
24                                 Hash(Succ(hop), seed))
25                 else nil
26             else nil
27         else nil
28     else nil
```

Every node n receiving a RREQ verifies that the alleged source *src* indeed signed the message with its private key (ll. 18–19), where the fields *hop* and *hash* are omitted as they are mutable fields; n aborts if this check fails. Node n verifies that $h^{maxhop-hop}(hash)$ and *tophash* ($= h^{maxhop}(seed)$) are equal. This is only the case if $hash = h^{hop}(seed)$, and thus in our modelling we can first "magically" obtain *seed* from *tophash* and then verify that $hash = \mathsf{Hash}(hop, seed)$ (ll. 20–21). Once again, n aborts if this check fails. Otherwise, n makes an entry into its routing table to note the distance to *src* (l. 22). Node n finally rebroadcasts the RREQ, after incrementing the *hop* field and hashing *hash* once (ll. 23-24).

Analysis. Let the following scenario be defined:

```
30 network N1(n) ≝
31     n[ReceiveRREQ(n), Keys(n1, PubKey(n1))∪
32         Keys(n2, PubKey(n2)) ∪ Keys(n3, PubKey(n3))]
33
34 network N2(n, dst) ≝
35     n[SendRREQ(n, dst) | ReceiveRREQ(n), Keys(n1, PubKey(n1))∪
36         Keys(n2, PubKey(n2)) ∪ Keys(n3, PubKey(n3))]
37
38 network μSAODV_RD_HC ≝ N2(n1, n3) || N1(n2) || N1(n3)
```

The scenario for μSAODV_RD_HC thus consists of three nodes n_1, n_2, and n_3 which each have the ability to receive RREQs, but only n_1 might actually initiate such a request. As before, the nodes are aware of the public keys of all honest nodes in the system. The attacker is once again taken as $N_A(1, n_A)$, and the abstract connectivity graph used is depicted in Figure 6.3.

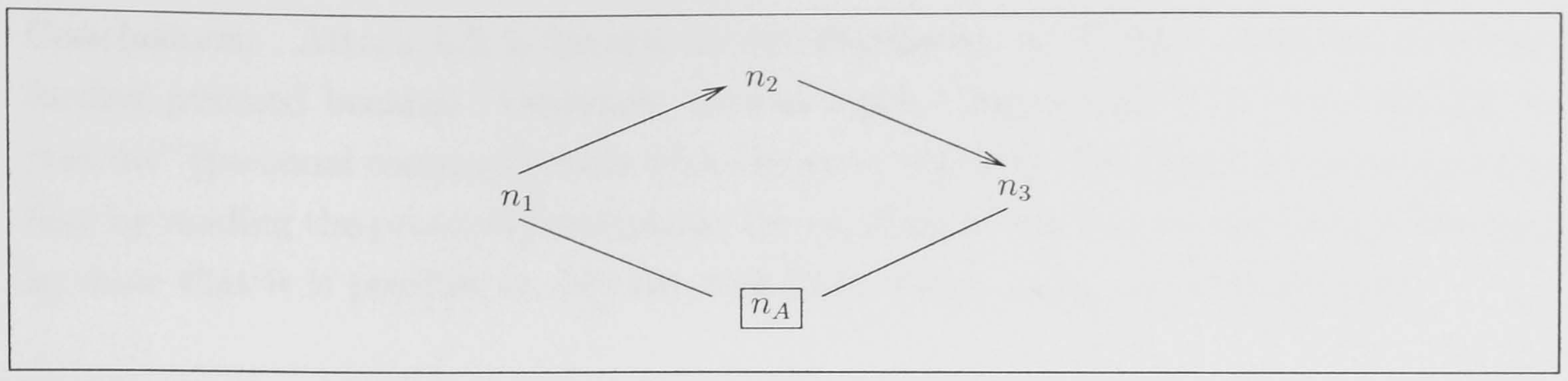

Figure 6.3: μSAODV_RD_HC: abstract connectivity graph

We propose to use the following consistency mediator:

$$\mu_n^{\widehat{G}}(U) = \begin{cases} \varepsilon & \text{if for } U = \mathsf{Rte}(dst, hop) \text{ there exist} \\ & \text{locations } n_1, \ldots, n_{hop+2} \text{ such that } n_1 = dst,\ n_{hop+2} = n \text{ and} \\ & \forall\, i \in \{1, \ldots, k-1\}.\ (n_i, n_{i+1}) \in \widehat{G} \\ U & \text{otherwise} \end{cases}$$

According to this definition, the length of a forward path from a destination to the current node has to be correctly estimated in the route table entry. We have the following analysis results for node n_3:

$$\begin{aligned} \sigma_{n_3}^{(1)} &= \{\mathsf{Rte}(n_1, hop) \ :\ hop \in \{1, 2, 3, \ldots\}\} \\ \sigma_{n_3}^{(2)} &= \{\mathsf{Rte}(n_1, hop) \ :\ hop \in \{0, 1, 2, \ldots\}\} \end{aligned}$$

The results indicate that network μSAODV_RD_HC might not be topology consistent, because the store of n_3 may hold a routing table entry which states that n_1 is reachable with 0 intermediate hops. This can be confirmed as follows:

Attack 6.2 (Wormhole) The attacker is connected to n_1 and thus comes into possession of the seed at the time of the initial transmission of the RREQ. Since neither seed nor hop count are signed, these can be modified freely by the attacker, in particular hop count 0 can be used. Node n_3 is in the end believed to be a direct neighbour of n_1 and will prefer this route to all other routes found. In a more elaborate scenario, two colluding attackers can exploit a private connection (wormhole; see Figure 6.4 and §1.1.4) to distribute the seed, such that n_{A_2} (n_{A_1}) can pretend to be n_1 (n_k). ⊠

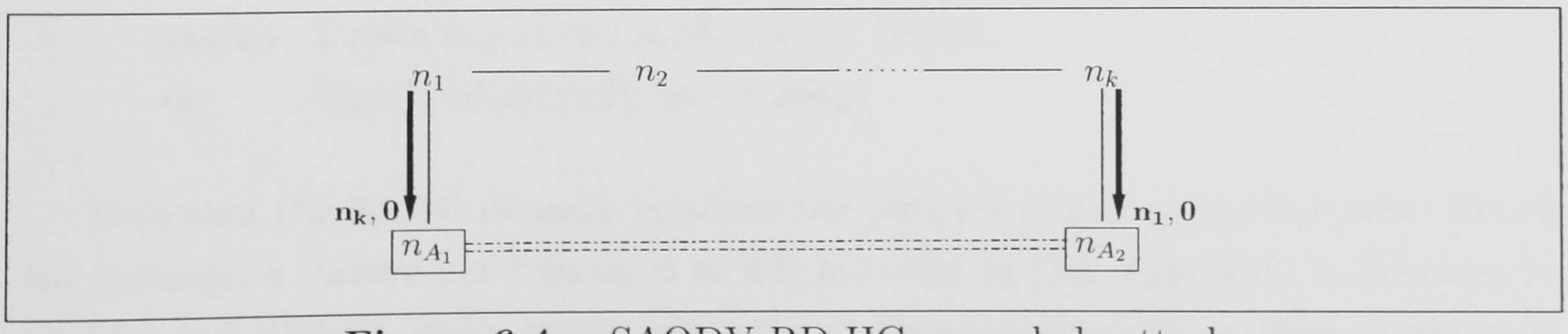

Figure 6.4: μSAODV_RD_HC: wormhole attack

Conclusions. Attack 6.2 is known to the developers of SAODV, but has not been further pursued because "wormhole attacks against hop count[s] are very difficult to prevent" (personal communication with Guerrero Zapata). We have first discovered the flaw by reading the protocol description; the analysis in this section was then performed to show that it is possible to discover this kind of flaw using our analysis only.

6.1.3 Route Maintenance

This section concerns route maintenance in AODV and how SAODV proposes to secure it. A node in AODV originates a *route error (RERR)* message either if it detects a link break for the next hop of a route while transmitting data (by way of link layer acknowledgements) or if it gets a data packet destined to a node for which it does not have a route. SAODV uses again digital signatures to secure the message as in §6.1.1, with the difference that every intermediate node signs the message.

Operation and Modelling. We model the route maintenance step for the case that nodes are not able to forward data packets because of missing routes; link layer acknowledgements are not modelled, but instead we may give certain nodes the ability to spontaneously emit a RERR message (as if in response to a link failure). Protocol μSAODV_RM handles two kinds of messages: DATA for sending data packets and RERR for route error messages.

```
1 signature DATA(dst : τ_n, addr : τ_n, snd : τ_n, data : τ_dat) : τ_data message
2 signature RERR(unr : τ_n, pubkey : τ_pubkey, snd : τ_n, sig : τ_sign) : τ_rerr message
```

The sending of data packets is not part of either AODV or SAODV but added in our modelling to yield a realistic scenario. The message components have the following meanings:

dst	IP address of the destination of the data packet
snd	IP address of the sender of the message (part of IP header)
addr	IP address of the addressee of the message (part of IP header)
data	Data packet
unr	IP address of the destination that is unreachable
pubkey	Public key of the sender of the RERR
sig	Sign(PrivKey(*snd*), *unr*, *pubkey*)

Note that the RERR message contains the public key *pubkey* of the sender *snd* of the message, a feature not contained in [61] but only in [59]. Also that, in contrast to RREQs and RREPs, *every sender* signs the contents of the route request packet (but again not the IP header part).

```
3 signature TT() : τ_bool twoway
4 signature FF() : τ_bool twoway
```

Boolean values are represented by the type τ_{bool} which has two values TT() and FF() representing true and false, respectively. We use Boolean values to model flags in route table entries.

```
8 signature Rte(τ_bool, τ_n, τ_n) : τ_rte secret
9 signature Dat() : τ_dat twoway
```

The route table entries have now the format Rte($valid, dst, next$) where $valid$ is a Boolean value expressing whether a route is considered valid or not (routes can become invalidated as the result of a RERR), and dst and $next$ represent destination and next hop as usual. Dat() symbolises a simple (empty) datum.

```
11 process SendDATA(n, dst) ≝
12     read Rte(TT(), dst; next)
13         out DATA(dst, next, n, Dat())
14     else nil
```

Every node n wanting to send a datum Dat() to a destination dst checks whether it possesses a valid route to dst (l. 12). If yes, the datum is sent by way of a data packet addressed to dst with sender n and next hop $next$ (l. 13). If no, the node terminates (l. 14). (More realistically, a RREQ would be initiated using the specification of §6.1.1).

```
16 process ReceiveDATA(n) ≝
17     in DATA(; dst, addr, snd, data).
18     if addr = n then
19         if dst = n then
20             store data
21         else
22             read Rte(TT(), dst; next)
23                 out DATA(dst, next, n, data)
24             else
25                 read Rte(FF(), dst; next)
26                     SendRERR(n, dst)
27                 else nil
28     else nil
```

On reception of a data packet, every node n checks whether it is the addressee; i.e. $addr = n$; n aborts if this check fails (l. 28). Otherwise, n checks whether it is the destination, i.e. $dst = n$. If yes, it stores the datum (l. 20). If no, n checks whether it possesses a valid route to dst (l. 22). If such a route exists, it resends the data packet,

now addressed to the next hop *next* and with sender n (l. 23). If the route is invalid instead (l. 25), n issues a RERR message in its own name with *dst* as the unreachable destination (l. 26).

```
30 process SendRERR(n, unr) ≝
31     out RERR(unr, PubKey(n), n, Sign(PrivKey(n), unr, PubKey(n)))
```

A node issues a route error by broadcasting a RERR message of correct format.

```
33 process ReceiveRERR(n) ≝
34     in RERR(; unr, pubkey, snd, sig).
35     checksig sig pubkey [unr, pubkey]
36         out RERR(unr, PubKey(n), n, Sign(PrivKey(n), unr, PubKey(n))).
37         read Rte(TT(), unr, snd; )
38             store Rte(FF(), unr, snd)
39         else nil
40     else nil
```

Every node n receiving a RERR verifies using the sent public key *pubkey* that the signature *sig* signs both the unreachable destination *unr* and the *pubkey* itself (l. 35); n aborts if this check fails. Otherwise it rebroadcasts the RERR concerning *unr*, setting the public key field to its own public key and the sender to n and appending a valid signature of the new message. Node n then checks whether it has a valid route for *unr* via the sender *snd* of the RERR (l. 37). If yes, n invalidates the route (l. 38), otherwise it terminates.

Analysis. Let the following scenario be defined:

```
42 process P(n) ≝ ReceiveDATA(n) | ReceiveRERR(n)
43
44 network μSAODV_RM ≝
45     n1[P(n1) | SendDATA(n1, n4), Rte(TT(), n4, n2) ∪ Rte(TT(), n6, n2)]
46   || n2[P(n2), Rte(TT(), n4, n3) ∪ Rte(TT(), n1, n1) ∪ Rte(TT(), n6, n5)]
47   || n3[P(n3), Rte(TT(), n4, n4) ∪ Rte(TT(), n1, n2)]
48   || n4[P(n4), Rte(TT(), n1, n3)]
49   || n5[P(n5) | SendRERR(n5, n6), Rte(TT(), n1, n2)]
```

The scenario for μSAODV_RM thus consists of five nodes which have a particular store set-up, which can best be described in conjunction with the assumed abstract connectivity graph displayed in Figure 6.5: there exists a bidirectional, topology consistent route between n_1 and n_4, and there had been a route between n_1 and n_6 which has recently broken between n_5 and n_6. Node n_5 is about to notify the network of the broken link by an RERR (l. 49). Node n_1 on the other hand is about to send a data

packet to n_4 (l. 45). All nodes are ready to process data packets and RERRs.

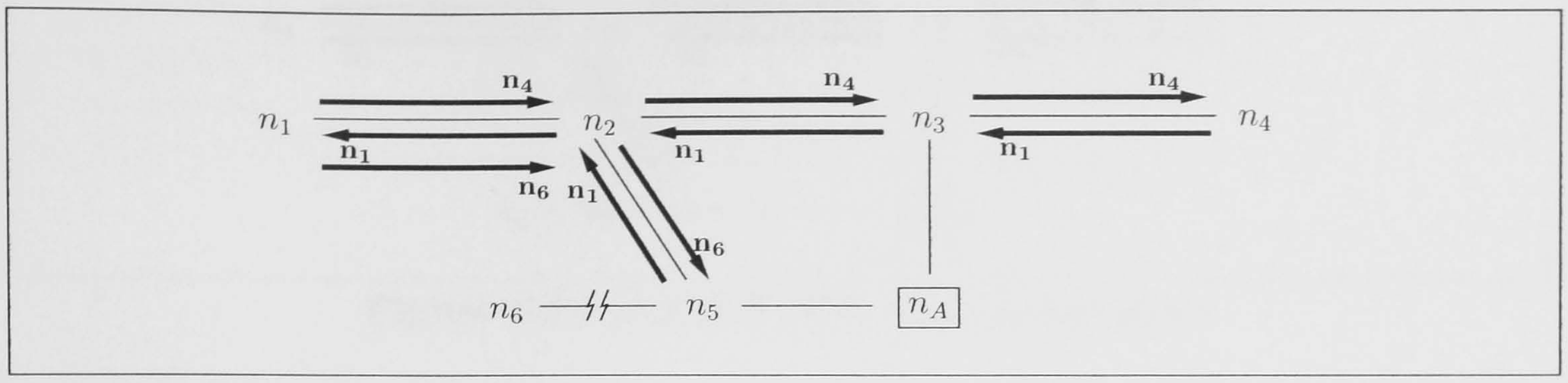

Figure 6.5: μSAODV_RM: network connectivity graph and routing table entries

Furthermore, we define the parametric attacker to be $A(1, \mathsf{PrivKey}(n_A) \cup \mathsf{PubKey}(n_A))$, thus having generated a key pair for itself which is not currently accepted in the network. We intend to use the following mediator:

$$\mu_n^{\widehat{G}}(U) = \begin{cases} \varepsilon & \text{if for } U = \mathsf{Rte}(\mathsf{FF}(), dst, next) \text{ there do } not \text{ exist} \\ & \text{locations } n_1 = n, n_2 = next, \ldots, n_k \text{ such that } n_k = dst \text{ such that} \\ & \forall\, i \in \{1, \ldots, k-1\}.\ (n_i, n_{i+1}) \in \widehat{G} \\ U & \text{otherwise} \end{cases}$$

Thus, invalidations of existing routes are tolerated if these routes had contradicted the network topology. The analysis yields the following results:

$$\begin{aligned}
\sigma_{n_1} &= \{\, \mathsf{Rte}(\mathsf{TT}(), n_6, n_2)\ ,\ \mathsf{Rte}(\mathsf{TT}(), n_4, n_2)\ ,\ \mathsf{Rte}(\mathsf{FF}(), n_4, n_2),\ \mathsf{Rte}(\mathsf{FF}(), n_6, n_2)\ \} \\
\sigma_{n_2} &= \{\, \mathsf{Rte}(\mathsf{TT}(), n_6, n_5)\ ,\ \mathsf{Rte}(\mathsf{TT}(), n_1, n_1)\ ,\ \mathsf{Rte}(\mathsf{TT}(), n_4, n_3)\ ,\ \mathsf{Rte}(\mathsf{FF}(), n_4, n_3), \\
&\quad\ \mathsf{Rte}(\mathsf{FF}(), n_6, n_5)\ \} \\
\sigma_{n_3} &= \{\mathsf{Rte}(\mathsf{FF}(), n_4, n_4),\ \mathsf{Rte}(\mathsf{TT}(), n_1, n_2)\ ,\ \mathsf{Rte}(\mathsf{TT}(), n_4, n_4)\ \} \\
\sigma_{n_4} &= \{\, \mathsf{Rte}(\mathsf{TT}(), n_1, n_3)\ \} \\
\sigma_{n_5} &= \{\, \mathsf{Rte}(\mathsf{TT}(), n_1, n_2)\ \}
\end{aligned}$$

The highlighted components are the ones present when analysing the network without attacker. The results indicate that the system might not be topology consistent since the attacker may invalidate a valid routing table entry $\mathsf{Rte}(\mathsf{TT}(), n_4, n_2)$. This is confirmed by the following attack:

Attack 6.3 The attacker may sign messages with its own key pair. As the sender of the message is never signed, nodes do not have the possibility to match the public key with the sender of the message. Thus, the attacker could use the process

$$\textbf{out}\ \mathsf{RERR}(n_4, \mathsf{PubKey}(n_A), n_4, \mathsf{Sign}(\mathsf{PrivKey}(n_A), n_4, \mathsf{PubKey}(n_A))).\ \textbf{nil}$$

in order to make n_3 invalidate its part of the route from n_1 to n_4. Eventually the entire route is invalidated as shown in Figure 6.6. ⊠

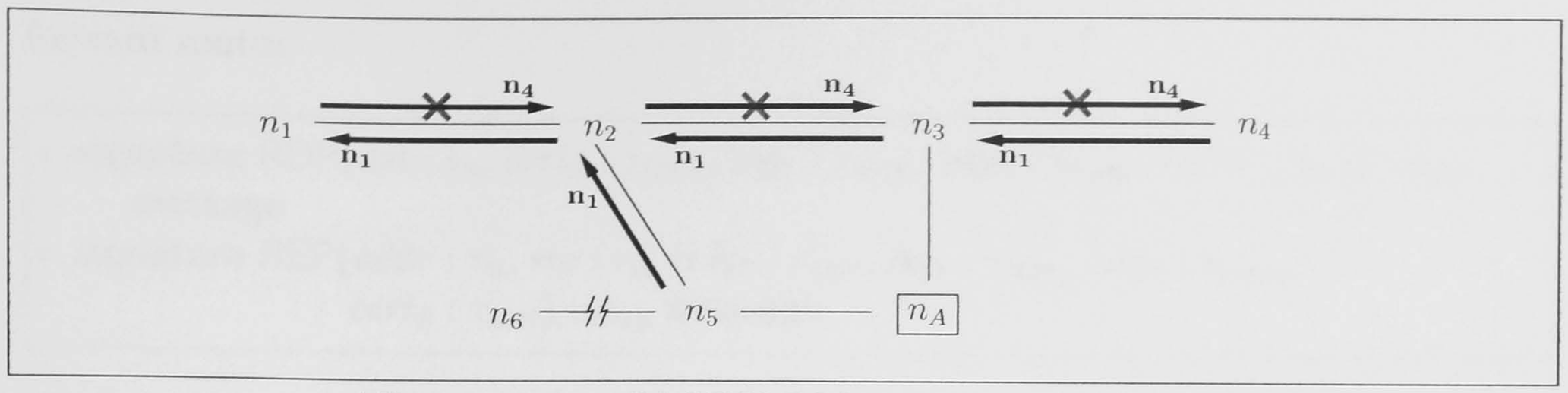

Figure 6.6: μSAODV_RM: route invalidations

Conclusions. Attack 6.3 has not been documented before to the best of our knowledge and was discovered by applying our analysis. The main reason for it to exist is the fact that in the latest draft of SAODV [59], the key management system *Simple Ad-Hoc Key Management (SAKM)* [60] has been marked as an optional component of SAODV only. If it is used, node IP addresses can be derived from a hash of the public key of the node, thus the public key could possibly be matched with the sender of the message and the flaw be eliminated (many details are missing in the specification document). We have reported this on the IETF MANET mailing list, and the developer Guerrero Zapata has subsequently stated his intention to either make the key management scheme mandatory or state more explicitly what properties a key management system has to fulfil in order to be applicable (the ability to derive IP addresses from public keys seems to be rather non-standard).

6.2 ARAN

Authenticated Routing for Ad-Hoc Networks (ARAN) [126] is a hop-by-hop routing protocol for mobile ad-hoc networks with built-in security features. In contrast to SAODV it is a stand-alone protocol rather then an extension to AODV; nevertheless there are many resemblances to AODV operation (one of the co-authors of [126] has been involved in the development of AODV). The main security mechanisms ARAN are based upon are *digital signatures* and *certification* using a trusted server. Similar to our approach in §6.1 we describe route discovery (§6.2.1) and route maintenance (§6.2.2) as two separate protocol parts.

6.2.1 Route Discovery

We define and analyse μARAN_RD, a model of the route discovery part of the protocol ARAN.

Operation and Modelling. The basic functionality of a route discovery step is analogous to AODV: *src* sends a *route discovery packet (RDP)* to *dst*, *dst* replies with a *reply packet (REP)* and these messages are used by intermediate nodes to set up reverse and

forward routes.

```
1 signature RDP(dst : τ_n, cert_1 : τ_cert1, sig_1 : τ_sign1, sig_2 : τ_sign2, cert_2 : τ_cert) : τ_rdp
    message
2 signature REP(addr : τ_n, src : τ_n, cert_1 : τ_cert, sig_1 : τ_sign1, sig_2 : τ_sign2,
                cert_2 : τ_cert) : τ_rep message
```

The involved message components denote the following:

dst	IP address of the destination of the route discovery
src	IP address of the source of the route discovery
$cert_1$	Certificate for src/dst in RDP/REP, respectively
sig_1	$\mathsf{Sign}_1(\mathsf{PrivKey}(src/dst), dst/src, cert_1)$ in RDP/REP, respectively
$cert_2$	Certificate for the sender snd of the RDP or REP
sig_2	$\mathsf{Sign}_2(\mathsf{PrivKey}(snd), dst/src, cert_1, sig_1)$ in RDP/REP, respectively
$addr$	IP address of the addressee of the message (part of IP header)

RDP and REP have thus a similar structure, with the difference that RDP is always broadcast and REP always unicast (to $addr$). Furthermore, note that there are two types of signatures, sig_1 signed by an endpoint node and sig_2 by the respective sender. The binding between a node and its public key is realised by way of certificates of the following structure:

```
3 signature Cert(τ_n, τ_pubkey, τ_sign3) : τ_cert twoway
4 signature Certs(τ_cert) : τ_certs secret
```

A certificate $\mathsf{Cert}(n, \mathsf{PubKey}(n), \mathsf{Sign}_3(\mathsf{PrivKey}(srv), n, \mathsf{PubKey}(n)))$ consists of the IP address of a node n to be certified, its public key, and a signature of these components with the private key of a trusted server srv. ARAN assumes that each node receives its certificate securely by some unspecified mechanism before the protocol starts; this is expressed in our modelling by the fact that each node has its proper certificate in its "certificate store" Certs. The remaining functions are straightforward, in particular Rte is modelled as in §6.1.1.

```
12 process SendRDP(dst, n) ≝
13     read Certs(; cert)
14         out RDP(dst, cert, Sign_1(PrivKey(n), dst, cert), Sign_2(PrivKey(n), dst, cert,
15                 Sign_1(PrivKey(n), dst, cert)), cert)
16     else nil
```

A node n starts a route discovery for dst by sending a RDP of correct format (ll. 14–15). In order to do this, n retrieves its own certificate $cert$ from the store (l. 13). Note

that ARAN does not actually require the *initial* RDP to contain the signature and certificate of the sending node, because it is identical with the source node. We include this in μARAN_RD as it simplifies the message handling routine; we have experimented with a variant specification which introduces two different message formats without a change in the results.

```
18 process ReceiveRDP(n) ≝
19     in RDP(; dst, cert1, sig1, sig2, cert2).
20     case cert1 of Cert(; src, pubkey1, sigcert1)
21         checksig3 sigcert1 PubKey(srv) [src, pubkey1]
22             checksig1 sig1 pubkey1 [dst, cert1]
23                 case cert2 of Cert(; snd, pubkey2, sigcert2)
24                     checksig3 sigcert2 PubKey(srv) [snd, pubkey2]
25                         checksig2 sig2 pubkey2 [dst, cert1, sig1]
26                             store Rte(src, snd).
27                             read Certs(; cert)
28                                 if dst = n then
29                                     out REP(snd, src, cert, Sign1(PrivKey(n), src, cert),
30                                             Sign2(PrivKey(n), src, cert,
31                                                 Sign1(PrivKey(n), src, cert)), cert)
32                                 else
33                                     out RDP(dst, cert1, sig1,
34                                             Sign2(PrivKey(n), dst, cert1, sig1), cert)
35                             else nil
36                         else nil
37                     else nil
38                 else nil
39             else nil
40         else nil
41     else nil
```

Lines 18 to 41 show the handling routine for RDPs. Note that not all the signature checks we have included are made explicit in [126], but we assume that it is meant that all of these checks are made.

On reception of a RDP, every node n first performs the checks related to the first certificate-signature pair (ll. 20–22). Node n retrieves the public key $pubkey_1$ of the alleged source *src* from the certificate $cert_1$ (l. 20) and verifies that the trusted server *srv* signed the certificate (l. 21). Node n further checks, using the now verified public key $pubkey_1$ of *src*, that *src* signed the first two components *dst* and $cert_1$ of the message. The checks related to the second certificate-signature pair (ll. 23–25) are performed analogously, with the outcome that n is convinced that *snd* has signed the first three components *dst*, $cert_1$, and sig_1 of the message. Node n aborts if any of the mentioned checks fails. Otherwise, n makes an entry into its routing table to provide a reverse route to *src* (l. 26). Node n then checks whether it is the destination of the RDP, i.e.

$dst = n$. If yes, n sends a REP addressed to the sender snd, doubly signed, and with n's own certificate $cert$ appended. If no, n removes the certificate and signature of snd from the original message, computes an own signature and appends it with $cert$ to the RDP before rebroadcasting.

The handling routine for REPs (see §B.2.1, ll. 43–66) is similar to that of RDPs and we will just point out the differences. In line 44, note that n makes sure only to process REPs which are addressed to it. After the checks (ll. 45–50), n makes an entry into its routing table to provide a forward route leading to dst (l. 51). If n was the source of the RDP in the first place, i.e. $src = n$, it terminates because the route is established (l. 54). Otherwise, n retrieves the next hop $next$ on the reverse route to src from its routing table and resends the REP addressed to $next$, removing the certificate and signature of snd from the original message and appending n's own (l. 57–58).

Analysis. Let the following scenario be defined, where we use the following abbreviation:

$$\mathsf{C}_n = \mathsf{Cert}(n, \mathsf{PubKey}(n), \mathsf{Sign}_3(\mathsf{PrivKey}(srv), n, \mathsf{PubKey}(n)))$$

68 **network** μARAN_RD $\stackrel{\text{def}}{=}$
69 $\quad n_1[SendRDP(n_3, n_1) \mid ReceiveREP(n_1), \mathsf{Certs}(\mathsf{C}_{n_1})]$
70 $\quad \| \; n_2[ReceiveRDP(n_2) \mid ReceiveREP(n_2), \mathsf{Certs}(\mathsf{C}_{n_2})]$
71 $\quad \| \; n_3[ReceiveRDP(n_3), \mathsf{Certs}(\mathsf{C}_{n_3})]$

The network topology and the attacker are identical to the ones used in the analysis of μSAODV_RD_DS.

The analysis results yield the following:

$$\begin{aligned}
\sigma_{n_1} &= \{\mathsf{Rte}(n_3, n_1), \mathsf{Rte}(n_3, n_2), \mathsf{Rte}(n_3, n_3), \mathsf{Rte}(n_1, n_1), \mathsf{Rte}(n_1, n_2), \mathsf{Rte}(n_1, n_3)\} \\
\sigma_{n_2} &= \{\mathsf{Rte}(n_1, n_1), \mathsf{Rte}(n_1, n_3), \mathsf{Rte}(n_3, n_1), \mathsf{Rte}(n_3, n_3)\} \\
\sigma_{n_3} &= \{\mathsf{Rte}(n_1, n_1), \mathsf{Rte}(n_1, n_2), \mathsf{Rte}(n_1, n_3), \mathsf{Rte}(n_3, n_1), \mathsf{Rte}(n_3, n_2), \mathsf{Rte}(n_3, n_3)\}
\end{aligned}$$

The terms $\mathsf{Rte}(n_3, n_3)$ at n_1 and $\mathsf{Rte}(n_1, n_1)$ at n_3 render the network topology inconsistent when using the consistency mediator of Example 5.9: nodes n_1 and n_3 are believed to be directly connected to each other, although they are not. Note however that it is no longer possible for an attacker to make route table entries point to arbitrary next hops.

Conclusion. The topology inconsistency of μARAN_RD is due to the possibility that the attacker may act as a relay, purely forwarding a message from n_1 to n_3. While this behaviour may sometimes be regarded as hard to defend against and the resulting problem as a weak form of topology inconsistency only, it is important to note that Protocol 4 of §5.3.1 shows a way to completely eliminate inconsistencies. This flaw was

discovered using our analysis, and, to the best of our knowledge, has not been described before.

6.2.2 Route Maintenance

Being an on-demand protocol, ARAN offers route maintenance when data is travelling on inactive routes or when links are broken. Similar to SAODV in §6.1.3, ARAN secures this mechanism by hop-by-hop signatures, but there are subtle differences which decide about the topology consistency of the protocols.

Operation and Modelling. In our model μARAN_RM we deal with two kinds of messages: the *error packet (ERR)*, part of ARAN's protocol definition, and an additional DATA packet to model the triggering of ERR messages.

```
1 signature DATA(addr : τ_n, src : τ_n, dst : τ_n, data : τ_dat) : τ_data message
2 signature ERR(addr : τ_n, src : τ_n, dst : τ_n, cert : τ_cert, sig : τ_sign1) : τ_err message
```

The message components have the following meanings:

addr	IP address of the addressee of the message (part of IP header)
src	IP address of the source of a route
dst	IP address of the destination of a route
data	Data packet
cert	Certificate for the sender *snd* of an ERR message
sig	$\mathsf{Sign}_1(\mathsf{PrivKey}(snd), src, dst, cert)$

Note that *src* and *dst* are source and destination of a route, but not necessarily of the message. For instance, a sender might recognise an expiry of the route between *src* and *dst*, and then send an ERR message with destination *src* to notify of this fact. The meaning of the remaining function signatures (see §B.2.2, ll. 3–12) is straightforwardly obtained from the analogous definitions of §6.1.3 and §6.2.1.

```
14 process SendDATA(n, dst) ≝
15     read Rte(TT(), dst; next)
16         out DATA(next, n, dst, Dat())
17     else nil
```

Every node n wanting to send a datum Dat() to a destination *dst* checks whether it possesses a valid route to *dst* (l. 15). If yes, the datum is sent by way of a data packet addressed to *dst* with sender n and next hop *next* (l. 16). If no, the node terminates (l. 17). (More realistically, a RDP would be initiated using the specification of §6.2.1).

```
19 process ReceiveDATA(n) ≝
20     in DATA(n; src, dst, data).
21     if dst = n then
22         store data
23     else
24         read Rte(TT(), dst; next)
25             out DATA(next, src, dst, data)
26         else
27             read Rte(FF(), dst; next_1)
28                 read Rte(TT(), src; next_2)
29                     read Certs(; cert)
30                         out ERR(next_2, src, dst, cert, Sign_1(PrivKey(n), src, dst, cert))
31                     else nil
32                 else nil
33             else nil
```

On reception of a data packet, every node n automatically filters the messages to receive only the ones addressed to itself (l. 20). Node n checks whether it is the destination, i.e. $dst = n$. If yes, it stores the datum (l. 22). If no, n checks whether it possesses a valid route to dst (l. 24). If such a route exists, it resends the data packet, now addressed to the next hop $next$ (l. 25). If instead the route to dst is invalid (l. 27), but there is a valid route to the source src (l. 28), n sends an ERR message to the (upstream) next hop $next_2$, marking the route src–dst as invalid (l. 30). In all other cases n terminates. (More elaborate handling could be imagined, but are not conclusive from the ARAN specification).

```
35 process ReceiveERR(n) ≝
36     in ERR(n; src, dst, cert, sig).
37     case cert of Cert(; snd, pubkey, sigcert)
38         checksig_3 sigcert PubKey(srv) [snd, pubkey]
39             checksig_1 sig pubkey [src, dst, cert]
40                 read Rte(TT(), dst, snd; )
41                     store Rte(FF(), dst, snd).
42                     read Rte(TT(), src; next)
43                         read Certs(; owncert)
44                             out ERR(next, src, dst, owncert,
45                                     Sign_1(PrivKey(n), src, dst, owncert))
46                         else nil
47                     else nil
48                 else nil
49             else nil
50         else nil
51     else nil
```

On reception of an ERR packet, every node n automatically filters the messages to receive only the ones addressed to itself (l. 36). Node n then performs the checks related to the certificate-signature pair (ll. 37–39). Node n retrieves the public key *pubkey* of the alleged sender *snd* from the certificate *cert* (l. 37) and verifies that the trusted server *srv* signed the certificate (l. 38). Node n further checks, using the now verified public key *pubkey* of *snd*, that *snd* signed the components *src*, *dst*, and *cert* of the message. Node n aborts if any of the mentioned checks fails. Otherwise n checks whether it has a valid route to *dst* via *snd* (l. 40) and aborts if no. If yes, it invalidates the route (l. 41) and additionally rebroadcasts the ERR message to the (upstream) next hop *next*, in case it exists, appending n's certificate *owncert* and a new signature (l. 44–45).

Analysis. Let the following scenario be defined:

53 **process** $P(n) \stackrel{\text{def}}{=} ReceiveDATA(n) \mid ReceiveERR(n)$
54
55 **network** μARAN_RM $\stackrel{\text{def}}{=}$
56 $\quad n_1[P(n_1) \mid SendDATA(n_1, n_3), \mathsf{Rte}(\mathsf{TT}(), n_3, n_2) \cup \mathsf{Certs}(\mathsf{C}_{n_1})]$
57 $\quad \| \; n_2[P(n_2), \mathsf{Rte}(\mathsf{TT}(), n_3, n_3) \cup \mathsf{Rte}(\mathsf{TT}(), n_1, n_1) \cup \mathsf{Certs}(\mathsf{C}_{n_2})]$
58 $\quad \| \; n_3[P(n_3), \mathsf{Rte}(\mathsf{TT}(), n_1, n_2) \cup \mathsf{Certs}(\mathsf{C}_{n_3})]$

The abstract connectivity graph is depicted in Figure 6.7 and shows the initial routing table entries as well. The definitions of attacker and consistency mediator are as in §6.1.3.

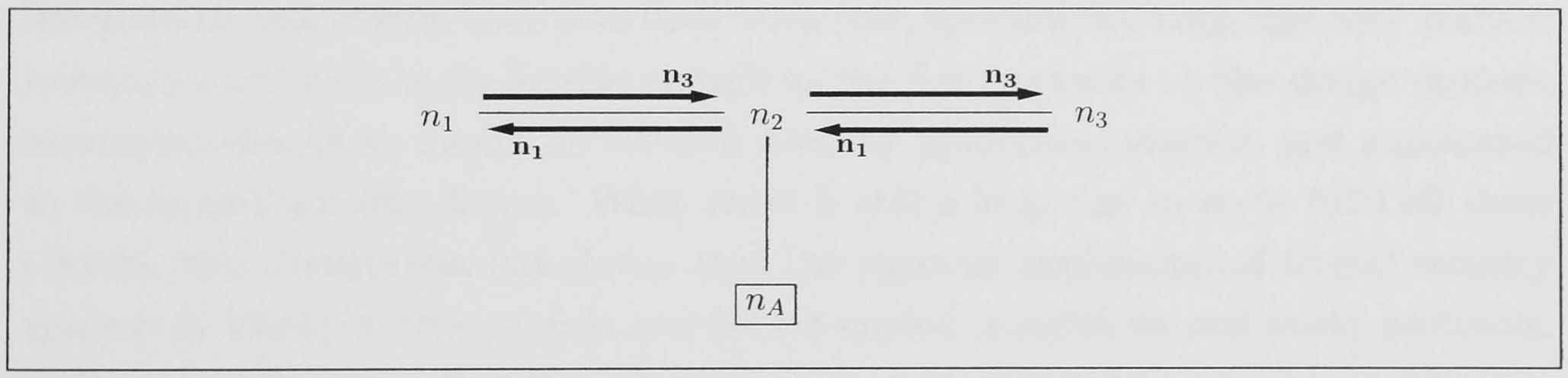

Figure 6.7: μARAN_RM: network connectivity graph and routing table entries

An application of the (untyped) finite analysis yields the following analysis estimates:

$$\begin{aligned}
\sigma_{n_1} &= \{\, \mathsf{Rte}(\mathsf{TT}(), n_3, n_2) \,\} \\
\sigma_{n_2} &= \{\, \mathsf{Rte}(\mathsf{TT}(), n_1, n_1) \,,\, \mathsf{Rte}(\mathsf{TT}(), n_3, n_3) \,\} \\
\sigma_{n_3} &= \{\, \mathsf{Rte}(\mathsf{TT}(), n_1, n_2) \,\}
\end{aligned}$$

It is therefore easy to see that μARAN_RM is topology consistent: the attacker cannot succeed in invalidating either of the routes present in the system.

Chapter 7

Conclusion

The belief that formal specification and analysis are beneficial for the development of secure systems can be considered widely accepted today. To a large degree, this can be attributed to the progress made on the development of verification frameworks for security protocols, an area which has received focused research attention over nearly three decades and has helped to uncover many protocol flaws. However, the rigorous application of formal techniques for security has not yet found its way into many other communities. One of these communities, namely the developers of communication protocols, we have focused on in this dissertation. Their development traditions seem to defy the application of formal techniques: their specification documents are written in prose, exhibit a great level of technical detail, and security is seldom a primary concern. Nevertheless, the rate at which threats appear and new protocol flaws are discovered calls for the application of verification tools. In order to convince protocol developers to integrate the use of such tools into their work flow, specification languages and analysis techniques will have to be flexible enough to put few restraints on the design process, expressive enough to model all relevant security properties, mature, and automated to the largest possible degree. While there is still a long way to go to fulfil all these criteria, this dissertation has shown that the rigorous application of formal security analysis is viable in this setting, can be automated, applied to real-world protocols, and, at the same time, that formal specifications can still remain readable and simple.

This final chapter is concerned with stocktaking and the question of how this work might best be continued and extended. In §7.1 we provide a review of the main research contributions of this dissertation. We conclude in §7.2 with a discussion of open problems and further directions of future work.

7.1 Review of Research Contributions

In this dissertation we have dealt with the question of whether classical formal specification and analysis techniques can be successfully applied to the security study of

communication protocols, beyond the thoroughly explored realm of security protocols. Considering the fact that communication protocols are a large and complex area, this venture has been helped by the choice of mobile ad-hoc networks and their protocols as a small and well-defined subclass for study. We found them particularly interesting because they are an area of very active research at the moment, with many protocols competing for standardisation. As such, they provide an excellent object for the study of design-time specifications. In contrast, many other protocol classes may be considered legacy systems, allowing for post-hoc specification and analysis only. Also, protocols for mobile ad-hoc networks are comparatively small. This allowed us to present them within the text of a dissertation, although we still had to idealise and abstract away some of their features.

In comparison with security protocols, secure routing protocols for mobile ad-hoc networks have proved to have their own challenges. In general, it seems fair to say that they exhibit a simpler interaction structure, yet base themselves on more complex data formats and internal computation. A careful modelling of a network's environment conditions and the definition of a refined attacker model turns out to be essential to express their security. With respect to security properties they appear less rich and subtle: a single main condition based on their service aspect seems relevant. We believe that our work has contributed to clarifying these aspects of mobile ad-hoc networks and their security, which seemed to be not well understood in the literature, and to expose the differences in analysing either protocol class. In the remainder of this section we summarise further research contributions of this dissertation:

1. *Definition of a process calculus CBS$^\sharp$ for the faithful modelling of the behaviour exhibited by mobile ad-hoc networks.* Building on the foundations of the broadcast calculus CBS, it provides the following distinct features: separation of process connectivity and process actions via the notion of *network topologies*; *local broadcast* as the base communication model; explicit notion of a computational entity as pair of process and private store; and, the ability to express complex data by a free term algebra with functions and case distinction with matching.

2. *Definition of behavioural equivalences for CBS$^\sharp$.* With bisimilarity we have explored the applicability of a classical notion of process equivalence to CBS$^\sharp$. Barbed equivalence adapts the notion of π-calculus barbs in order to expose stored elements hidden by the reduction relation. The concept of *mediation* has been introduced to describe a parametrisation of equivalences with a function μ on terms, to the end that idealised systems can be expressed straightforwardly. This development has led to the notion of mediated equivalence which has proved to be suitable for security analysis.

3. *Definition of a Control Flow Analysis to overapproximate the behaviour of net-*

works described in CBS$^\sharp$. The analysis is specified concisely using the flow logic paradigm, and is proved correct with respect to the operational semantics of CBS$^\sharp$. The analysis is flexible as provable properties are not hardcoded but obtained by a μ-parametrised post-analysis step.

4. *Prototype implementation of the Control Flow Analysis.* The original analysis has been transformed using tree grammars and a labelling of CBS$^\sharp$ such that it can be executed over a finite universe of values. In this form, the analysis is guaranteed to terminate. The finite analysis has been proved correct with respect to the original analysis such that results obtained from the prototype still yield a behavioural overapproximation for networks. We have shown that the finite analysis can straightforwardly be converted into a generation algorithm for constraints specified in a Horn-like fragment of first-order predicate logic. In combination with an off-the-shelf constraint solving tool, we have thus obtained an automated analysis framework.

5. *Formalisation of a parametric attacker model.* We have discussed the fact that the classical Dolev-Yao model proves to be too powerful in the context of mobile ad-hoc networks. We have sketched a more suitable, parametrised model which relaxes the Needham-Schroeder notion of attacker omnipresence. We have shown that this attacker model can be expressed straightforwardly in CBS$^\sharp$, and have developed a simple technique to make sure that the model fulfils the requirements of the perfect cryptography assumption.

6. *Definition of a type discipline for attacker implementations.* We have analysed the trade-off between precision and complexity for the finite analysis. Limiting the state space arising from the analysis of the attacker turned out to be crucial. We have developed a type discipline for attacker implementations to address this problem.

7. *Definition of a parametric security condition for routing protocols.* We have analysed the problem of communication protocol security and have described it as the ability to provide the desired network service in an adversary environment. In the setting of routing protocols for mobile ad-hoc networks, we have defined a simple security condition based on this idea. The security condition is expressed via mediated equivalence for CBS$^\sharp$ and is parametric with respect to: the chosen attacker model; the network topology; and, the data format used by the protocol. The condition is fully integrated into the framework and can thus be derived automatically.

8. *Case studies of two real-world protocols for secure routing in mobile ad-hoc networks.* We have shown the usability of CBS$^\sharp$ as a specification language by apply-

ing it to the modelling of two real-world protocols. While the type discipline for attacker implementations has proved to be adequate, we have experienced that some protocol idealisations and separation of protocol parts are still indispensable to assure a tractable analysis complexity. Using the prototype implementation, we have been able to detect three distinct exploitable flaws in one of the protocols, and have detected omissions and ambiguities in both protocols.

7.2 Directions for Future Work

We believe that our dissertation has helped to identify mobile ad-hoc networks as an interesting and worthwhile application domain for rigorous formal security analysis. We thus hope that our work will induce future research efforts that leverage our approach and might eventually convince protocol developers to adopt the use of formal techniques. A variety of open questions has to be solved until then, some of which we describe in this section.

7.2.1 Quantitative Network Topology Abstractions

The notion of the network topology $\mathcal{T}$ allows for the modelling of link mobility in CBS$^\sharp$. It can be used to specify mobility for a given network in various ways, which range from fixed connectivity ($\mathcal{T}$ has cardinality one) to full mobility ($\mathcal{T}$ equals $\mathcal{T}_{max}$). While this flexibility might prove to be helpful or even necessary when proving particular properties by hand, it is of little significance when used in conjunction with our Control Flow Analysis or security condition: both the analysis and the security condition use the crude abstraction $\mathcal{G}(\mathcal{T})$ (see §4.2.1) which yields a single connectivity graph; a careful modelling of $\mathcal{T}$ is then no longer sensible as a fixed connectivity model could be specified from the outset.

The abstraction $\mathcal{G}(\mathcal{T})$ was motivated by two reasons. Firstly, we want to *overapproximate* the behaviour occurring in the network and thus have to ensure the following property so that all messages which *may* be received are considered by our analysis (compare Fact 4.3):

$$\exists\, G \in \mathcal{T}.\ (m,n) \in E(G) \Rightarrow (m,n) \in E(\mathcal{G}(\mathcal{T}))$$

Secondly, we understand security in routing networks as being defined by the tense relation between the state of the *actual* network connectivity and its *virtual* representation in the routing tables of nodes. We thus cannot reason about security in this setting if the actual network connectivity remains dynamic, i.e. undefined. Therefore, the security condition of §5.2.1 distinguishes only the situations in which nodes are *never* or *sometimes* connected. Luckily, even with such a coarse abstraction of mobility, we have been able to derive security results.

The first reason may be dismissed as a consequence of our choice of analysis technique, essentially a classical abstract interpretation. The second is however directly tied to our understanding of the security problem and justifies the approach. Nevertheless, one might wonder whether the extra design space offered by CBS$^\sharp$ through $\mathcal{T}$ might not be put to good use in other contexts. Even if we want to keep an abstract interpretation approach, this might be possible through the use of quantitative abstractions of $\mathcal{T}$ which retain more of its original information than the current two-valued abstraction.

For instance, it is possible to express in CBS$^\sharp$ that a network of four nodes may assume any of the connectivity configurations of Figure 7.1.

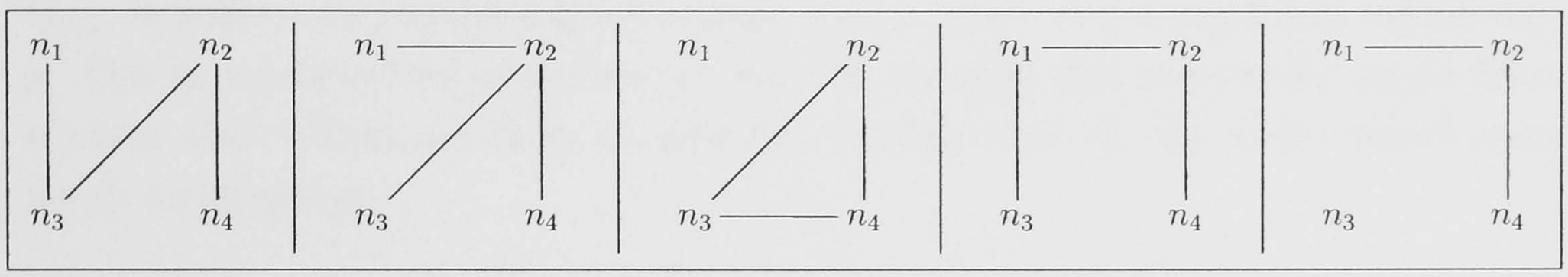

Figure 7.1: Example: network topology

Figure 7.2 (a) displays the two-valued abstraction of this scenario the control flow analysis of this network works with. More information could be preserved when using a quantitative abstraction which associates with every link the probability of its availability (assuming equal likelihood of any $T \in \mathcal{T}$), as shown in Figure 7.2 (b).

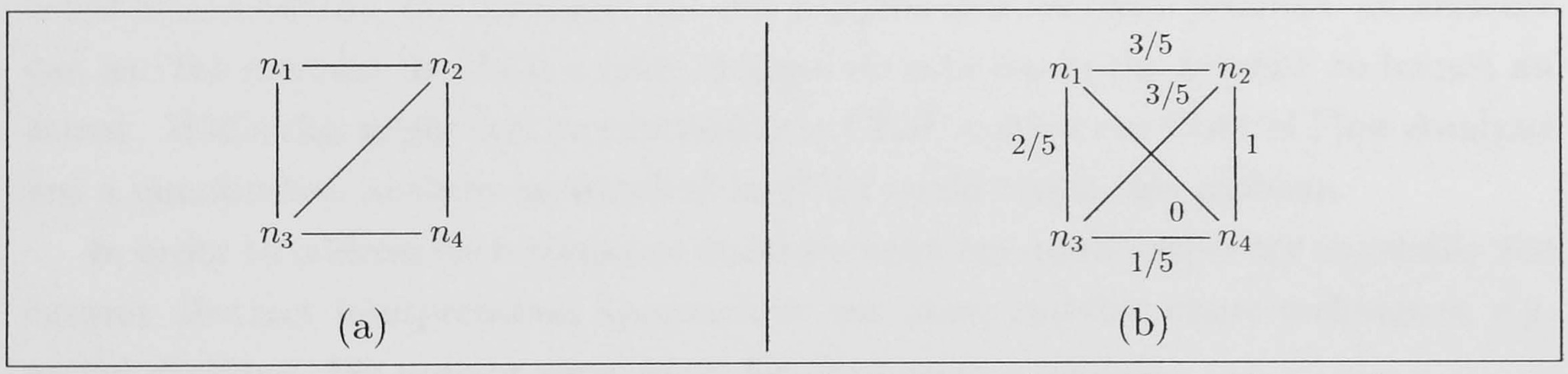

Figure 7.2: Two-valued and many-valued network topology abstractions

An advanced analysis can take this additional information into account in order to address questions like: What is the probability that a packet sent from n_3 is received at n_4 using any available route? In our framework this question can be answered by determining the probability that the following transition takes place:

$$n_3[\textbf{out}\ \mathsf{Data}(), \varepsilon] \parallel n_4[\textbf{in}\ x.\,\textbf{store}\ x, \varepsilon] \parallel \ldots \longrightarrow^*_{\mathcal{T}} n_3[\textbf{nil}, \varepsilon] \parallel n_4[\textbf{nil}, \mathsf{Data}()] \parallel \ldots$$

In turn, the ability to answer such questions may be used to address problems in mobile ad-hoc networks which require a quantitative treatment, e.g. efficiency (expected packet loss) or sensitivity of networks to attacks (which we have explored using a preliminary version of CBS$^\sharp$ in [98]). It remains an open question whether these kinds of analyses could be computed automatically, for example using the framework of *Probabilistic Abstract Interpretation (PAI)* [43] which recasts classical abstract interpretation

in a setting of linear spaces.

Furthermore, the following problem might become relevant: oftentimes an analyst will not have access to connectivity information as specific as the scenario depicted in Figure 7.1. Instead, it might be more straightforward to choose parameters which are easier to guess for a specific scenario, such as the size of the network, the size of the area the nodes occupy, or the average speed of a node. It would thus be interesting to determine the connectivity properties of ad-hoc networks with respect to such constraints. One of the earliest network models is that of a *random graph* by Erdös and Rényi [49], a graph generated by some random process. The most commonly studied model, called $G_{n,p}$, includes each possible edge of a graph with n nodes with independent probability p. This is unsatisfactory as it does not take into account that some nodes might form clusters while others are likely be isolated. Modifications of this model would seem worth investigating.

7.2.2 True Mobility

In the previous section we have argued that finer grained abstractions of the network topology allow to preserve more mobility information. Continuing this line of thought, one could also try to get rid of all mobility abstraction and analyse the dynamics of the networks directly. A benefit of this approach would be that very detailed scenarios could be considered. For example, one can imagine that, in some protocol, an attacker can use the moment in which a node changes its position in the network to launch an attack. While this might still be expressible in CBS$^\sharp$, neither our Control Flow Analysis nor a quantitative analysis as sketched in §7.2.1 could reveal this problem.

In order to address such scenarios in future work one could either try to modify the current abstract interpretation approach or use other suitable static techniques, e.g. model checking. We present some ideas for the former approach.

An analyst might suspect that a specific attack might be launched when a network N changes its connectivity graph from G_1 to G_2 at some point during its computation:

$$N \longrightarrow^*_{G_1} N_1 \xrightarrow{(U,m)!}_{G_2} N_2$$

Using $\widehat{G} = \mathcal{G}(\{G_1, G_2\})$, the current analysis

$$(\kappa, \sigma) \models_{\widehat{G}} N$$

cannot reveal anything about this particular question. Instead we could consider the two analyses

$$(\kappa_1, \sigma_1) \models_{G_1} N$$
$$(\kappa_2, \sigma_2) \models_{G_2} N \parallel M_1 \parallel M_2,$$

where:

$$M_1 \stackrel{\text{def}}{=} \|_{(T,n)\in\kappa_1} n[\textbf{out } T, \varepsilon]$$
$$M_2 \stackrel{\text{def}}{=} \|_{(T,n)\in\sigma_1} n[\textbf{store } T, \varepsilon].$$

It is easy to show, using the subject reduction theorem and the definitions of M_1 and M_2, that both the following judgements hold:

$$(\kappa_1, \sigma_1) \vDash_{G_1} N_1$$
$$(\kappa_2, \sigma_2) \vDash_{G_2} N_1$$

This provides information about the behaviour of network N_1 for the case that the connectivity change takes place or not. The set differences $\kappa_2 \backslash \kappa_1$ and $\sigma_2 \backslash \sigma_1$ could thus reveal potentially valuable information about the consequences of this change.

7.2.3 Cut-Offs for Network Topologies

Even when mobility is not considered, the *verification* of ad-hoc networks is much complicated by the infinite number of possible network configurations. In this dissertation, we have thus focused on falsification. This means that the analyst chooses a network configuration considered most likely to expose an attack, and runs the analysis for it. Either an attack is found, or the protocol proved correct *for this configuration.* However, to prove the protocol correct once and for all, *all* configurations would have to be considered. While it is still imaginable to analyse all network topologies for networks with only very few nodes, this brute force approach becomes intractable soon since there are $\binom{k}{2}$ possible configurations for networks with k nodes. One could thus ask whether there is an upper bound on the network size to be considered.

A brute force approach cannot reveal this upper bound: since our security condition is based on the notion of connectivity, the possibility exists that even if we have analysed all networks of size k_0, there might be an attack on networks with size $k_0 + 1$. In a more refined approach however, one could try to use the fact that these networks are homogeneous in the sense that all nodes run essentially the same process. Clarke et al. [35] have used this observation for model checking: they show how to decompose an arbitrary network graph into multiple constant size networks; model checking is then reduced from one call on a large graph to several calls on small networks; they generalise this to obtain *cut-offs*, i.e. reducing an infinite class of networks to a finite number of model checking calls. It would be interesting to see whether this idea can be applied to static analysis. Of course, in the context of our framework this seems to be very ambitious since the approach of [35] is helped by the fact that they consider very simple token-passing processes only.

Appendix A

Proofs

A.1 Semantic Correctness of the Control Flow Analysis

Lemma 4.7

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

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

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

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

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

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

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

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

$\widetilde{x} = x_{j+1}, \ldots, x_k$, $\widetilde{U} = U_{j+1}, \ldots, U_k$.

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

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

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

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

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

Case Refl-Red. Then $N = N'$. Nothing to show.

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

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

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

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

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

Case REFL-STRUCT. Then $N = N'$. Nothing to show.

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

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

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

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

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

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

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

Lemma 4.8 *If* $N \xrightarrow{(T,n)!}_{G \in \mathcal{T}} N'$ *and* $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} N$, *then* $(T, n) \in \kappa$.

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

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

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

Case STRUCT. We have $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} N$ by assumption. We assume the premises of STRUCT, in particular $N \equiv M$ and $M \xrightarrow{(T,n)!}_{G \in \mathcal{T}} M'$. Applying Lemma 4.7

(2) to $N \equiv M$ and $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} N$ gives $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} M$. Thus, the induction hypothesis can be applied to give $(T, n) \in \kappa$.

Case Red. Analogous to case Struct (after assuming the premises of Red, apply Lemma 4.7 (1) to $N \rightarrow M$ and $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} N$). □

Theorem 4.9 (Subject Reduction)

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

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

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

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

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

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

Case In$_1$. Then $M = n[\mathbf{in}\ x.P, S]$ and $M' = n[P[U/x], S]$. By assuming $N \xrightarrow{(U,m)!}_{G \in \mathcal{T}} N'$ and $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} N$, Lemma 4.8 gives $(U, m) \in \kappa$ $(*)$. From the preconditions of In$_1$ we know $(m, n) \in E(G)$, and with Fact 4.3 we have $(m, n) \in E(\mathcal{G}(\mathcal{T}))$ $(**)$.

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

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

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

Case PPAR. Then $M = M_1 \parallel M_2$ and $M' = M_1' \parallel M_2'$. We have $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} M_1 \parallel M_2$ by assumption, and with CFA-PPAR thus $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} M_1$ and $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} M_2$. 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 $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} M_1'$ and $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} M_2'$. With CFA-PPAR we have $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} M_1' \parallel M_2'$.

Case STRUCT. We have $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} M$ 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 4.7 (2) to $M \equiv L$ and $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} M$ we have $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} L$. Hence, we can apply the induction hypothesis on $L \xrightarrow{(U,m)\sharp}_{G \in \mathcal{T}} L'$ to have $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} L'$. By applying Lemma 4.7 (2) again, we have $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} L'$.

Case RED. Analogous to case STRUCT (after assuming the premises of RED, apply Lemma 4.7 (1) to $M \rightarrow L$ and $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} M$). □

A.2 Correctness of the Finite Analysis

Lemma 4.18 *Let $\theta^\star$ be given and define for all $x \in \mathcal{X}$*

$$\theta(x) \in \{U \ : \ \exists\, \ell \in \mathcal{L}.\ \theta^\star(x) = \ell \wedge U \in L(\gamma, \ell)\}.$$

Then the following implication holds:

$$\textit{If } \gamma \vDash^{\theta^\star} T^\ell \textit{ then } \lfloor T^\ell \rfloor \theta \in L(\gamma, lab^{\theta^\star}(T^\ell))$$

Proof. By structural induction on T^ℓ.

Case $T^\ell = n^\ell$. We have $\gamma \vDash^{\theta^\star} n^\ell$ by assumption, therefore $n \in \gamma(\ell)$ with rule FIN-NAME. With Definition 4.16 thus $n \in L(\gamma, \ell)$ which proves the case since $\lfloor n^\ell \rfloor \theta = n$.

Case $T^\ell = x$. We have $\lfloor x \rfloor \theta = \theta(x)$, and $\theta(x) \in L(\gamma, \theta^\star(x))$ by definition of $\theta(x)$.

Case $T^\ell = f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k})$. We have $\gamma \vDash^{\theta^\star} f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k})$ by assumption, therefore $\bigwedge_{i=1}^k \gamma \vDash^{\theta^\star} T_i^{\ell_i}$ $(*)$ and $f(lab^{\theta^\star}(T_1^{\ell_1}), \ldots, lab^{\theta^\star}(T_k^{\ell_k})) \in \gamma(\ell)$ with FIN-FUN. We can thus apply the induction hypothesis k times to $(*)$ to get $\bigwedge_{i=1}^k \lfloor T_i^{\ell_i} \rfloor \theta \in L(\gamma, lab^{\theta^\star}(T^{\ell_i}))$. Since $\lfloor f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k}) \rfloor \theta = f(\lfloor T_1^{\ell_1} \rfloor \theta, \ldots, \lfloor T_k^{\ell_k} \rfloor \theta)$, the result follows by Definition 4.16. □

Lemma 4.19 *Let $\theta^\star$ be given and let θ be defined like in Lemma 4.18. Then the following implication holds:*

$$\textit{If } \gamma \vDash^{\theta^\star} T^\ell \textit{ and } \lfloor T^\ell \rfloor \theta \in L(\gamma, \ell') \textit{ then } \mathcal{E}_\gamma^{\theta^\star}(\ell', T^\ell)$$

Proof. By structural induction on T^ℓ.

Case $T^\ell = n^\ell$. With Lemma 4.18, we have $n \in L(\gamma, \ell)$ from $\gamma \vDash^{\theta^\star} n^\ell$. We know by assumption that $\lfloor n^\ell \rfloor \theta = n \in L(\gamma, \ell')$. Therefore $L(\gamma, \ell') \cap L(\gamma, \ell) \neq \emptyset$ which gives $\mathcal{E}_\gamma^{\theta^\star}(\ell', n^\ell)$ with EQ-NAME.

Case $T^\ell = x$. With Lemma 4.18, we have $\theta(x) \in L(\gamma, lab^{\theta^\star}(x))$. We know by assumption that $\lfloor x \rfloor \theta = \theta(x) \in L(\gamma, \ell')$. Therefore $L(\gamma, \ell') \cap L(\gamma, lab^{\theta^\star}(x)) \neq \emptyset$ and the result follows with EQ-VAR.

Case $T^\ell = f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k})$. We have $\lfloor f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k}) \rfloor \theta = f(\lfloor T_1^{\ell_1} \rfloor \theta, \ldots, \lfloor T_k^{\ell_k} \rfloor \theta) \in L(\gamma, \ell')$ by assumption. Thus by Definition 4.16 there exist $\ell'_1, \ldots, \ell'_k$ such that $f(\ell'_1, \ldots, \ell'_k) \in \gamma(\ell')$ and $\bigwedge_{i=1}^k \lfloor T_1^{\ell_1} \rfloor \theta \in L(\gamma, \ell'_i)$ $(*)$. Furthermore, we have $\gamma \vDash^{\theta^\star} f^\ell(T_1^{\ell_1}, \ldots, T_k^{\ell_k})$ by assumption, therefore $\bigwedge_{i=1}^k \gamma \vDash^{\theta^\star} T_i^{\ell_i}$ $(**)$ with FIN-FUN. We can apply the induction hypothesis k times to $(**)$ and $(*)$ to have $\bigwedge_{i=1}^k \mathcal{E}_\gamma^{\theta^\star}(lab^{\theta^\star}(\ell'_i, T_i^{\ell_i})$. Application of EQ-FUN concludes the proof. □

Lemma 4.20 *Let $\kappa^\star$, $\sigma^\star$, and $\theta^\star$ be given and define*

$$\begin{array}{lcl} \kappa & = & \{(U, n) \,:\, \exists\, \ell \in \mathcal{L}.\, (\ell, n) \in \kappa^\star \wedge U \in L(\gamma, \ell)\} \\ \sigma & = & \{(U, n) \,:\, \exists\, \ell \in \mathcal{L}.\, (\ell, n) \in \sigma^\star \wedge U \in L(\gamma, \ell)\} \\ \theta(x) & \in & \{U \,:\, \exists\, \ell \in \mathcal{L}.\, \theta^\star(x) = \ell \wedge U \in L(\gamma, \ell)\} \text{ for all } x \in \mathcal{X}. \end{array}$$

Then the following implications hold:

(1) *If $\gamma \vDash^{\theta^\star} T^\ell$ and $(lab^{\theta^\star}(T^\ell), n) \in \kappa^\star$ then $(\lfloor T^\ell \rfloor \theta, n) \in \kappa$*

(2) *If $\gamma \vDash^{\theta^\star} T^\ell$ and $(lab^{\theta^\star}(T^\ell), n) \in \sigma^\star$ then $(\lfloor T^\ell \rfloor \theta, n) \in \sigma$*

Proof. We prove (1) only, (2) is completely analogous. We have $\gamma \vDash^{\theta^\star} T^\ell$ by assumption, therefore $\lfloor T^\ell \rfloor \theta \in L(\gamma, lab^{\theta^\star}(T^\ell))$ with Lemma 4.18. Since also $(lab^{\theta^\star}(T^\ell), n) \in \kappa^\star$, there exists an $\ell \in \mathcal{L}$ such that $(\ell, n) \in \kappa^\star \wedge \lfloor T^\ell \rfloor \theta \in L(\gamma, \ell)$. Therefore $(\lfloor T^\ell \rfloor \theta, n) \in \kappa$ by definition of κ. □

Theorem 4.21 *Let $\kappa^\star$, $\sigma^\star$, and $\theta^\star$ be given and let κ, σ, and θ be defined as in Lemma 4.20. Then the following implications hold:*

(1) *If $(\kappa^\star, \sigma^\star, \gamma) \vDash^{\theta^\star}_{\widehat{G}, n}$ labelling(P) then $(\kappa, \sigma) \vDash^{\theta}_{\mathcal{G}(\mathcal{T}), n} P$*

(2) *If $(\kappa^\star, \sigma^\star, \gamma) \vDash_{\widehat{G}}$ labelling(N) then $(\kappa, \sigma) \vDash_{\mathcal{G}(\mathcal{T})} N$*

Proof. **Ad** (1). By structural induction on labelled processes.

Case nil. Nothing to show.

Case out $T^\ell.P$**.** We have $(\kappa^\star, \sigma^\star, \gamma) \vDash^{\theta^\star}_{\widehat{G},n}$ **out** $T^\ell.P$ by assumption. Therefore we have $\gamma \vDash^{\theta^\star} T^\ell$, $(lab^{\theta^\star}(T^\ell), n) \in \kappa$, and $(\kappa^\star, \sigma^\star, \gamma) \vDash^{\theta^\star}_{\widehat{G},n} P$ by FIN-OUT. Thus by Lemma 4.20 (1) we have $(\lfloor T^\ell \rfloor \theta, n) = (T\theta, n) \in \kappa$ and by induction hypothesis we have $(\kappa, \sigma) \vDash^{\theta}_{\mathcal{G}(\mathcal{T}),n} P$. With CFA-OUT follows the result.

Case in $x.P$**.** By CFA-IN, we have to show

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

Thus fix $(U, m) \in \kappa$ and assume $(m, n) \in E(\mathcal{G}(\mathcal{T})$ $(*)$. By definition of κ, $\exists\, \ell \in \mathcal{L}.\ (\ell, m) \in \kappa^\star \wedge U \in L(\gamma, \ell)$ $(**)$. We have $(\kappa^\star, \sigma^\star, \gamma) \vDash^{\theta^\star}_{\widehat{G},n}$ **in** $x.P$ by assumption, therefore

$$\forall\, (\ell', m) \in \kappa.\ (m, n) \in E(\widehat{G}) \Rightarrow (\kappa^\star, \sigma^\star, \gamma) \vDash^{\theta^\star[x \mapsto \ell']}_{\widehat{G},n} P.$$

Thus for the pair (ℓ, m) which exists due to $(**)$ and considering $(*)$ we have $(\kappa^\star, \sigma^\star, \gamma) \vDash^{\theta^\star[x \mapsto \ell]}_{\widehat{G},n} P$. We can apply the induction hypothesis to have $(\kappa, \sigma) \vDash^{\theta[U/x]}_{\mathcal{G}(\mathcal{T}),n} P$ since $\theta[U/x](x) \in \{U : \theta^\star[x \mapsto \ell](x) = \ell \wedge U \in L(\gamma, \ell)\}$ where $U \in L(\gamma, \ell)$ because of $(**)$.

Case store $T^\ell.P$**.** Analogous to case **out** $T^\ell.P$. Apply Lemma 4.20 (2) instead of Lemma 4.20 (1).

Case read $x.P$**.** Analogous to case **in** $x.P$.

Case case T^ℓ **of** $f(T_1^{\ell_1}, \ldots, T_j^{\ell_j}; x_{j+1}, \ldots, x_k)\ P_1$ **else** P_2**.** We know by assumption and FIN-CASE:

(a) $\gamma \vDash^{\theta^\star} T^\ell$

(b) $\bigwedge_{i=1}^{j} \gamma \vDash^{\theta^\star} T_i^{\ell_i}$

(c) $\forall\ \ell'_{j+1}, \ldots, \ell'_k.\ (\exists\ \ell'_1, \ldots, \ell'_j.\ f(\ell'_1, \ldots, \ell'_k) \in \gamma(lab^{\theta^\star}(T^\ell)) \wedge \bigwedge_{i=1}^{j} \mathcal{E}^{\theta^\star}_{\gamma}(\ell'_i, T_i^{\ell_i})) \Rightarrow (\kappa^\star, \sigma^\star, \gamma) \vDash^{\theta^\star[x_i \mapsto \ell'_i]_{i=j+1}^{k}}_{\widehat{G},n} P_1$

(d) $(\kappa^\star, \sigma^\star, \gamma) \vDash^{\theta^\star}_{\widehat{G},n} P_2$

To show:

(i) $\lfloor T^\ell \rfloor \theta = f(U_1, \ldots, U_k) \wedge \bigwedge_{i=1}^{j} \lfloor T_i^{\ell_i} \rfloor \theta = U_i \Rightarrow (\kappa, \sigma) \vDash^{\theta[U_i/x_i]_{i=j+1}^{k}}_{\mathcal{G}(\mathcal{T}),n} P_1$

(ii) $(\kappa, \sigma) \vDash^{\theta}_{\mathcal{G}(\mathcal{T}),n} P_2$

Ad (i)**.** Assume

(e) $\lfloor T^\ell \rfloor \theta = f(U_1, \ldots, U_k)$ and

(f) $\bigwedge_{i=1}^{j} \lfloor T_i^{\ell_i} \rfloor \theta = U_i$

and try to show $(\kappa, \sigma) \vDash^{\theta[U_i/x_i]_{i=j+1}^k}_{\mathcal{G}(\mathcal{T}),n} P_1$. For this, under assumptions (a)–(f), consider the following intermediate goal:

There exist $\ell'_1, \ldots, \ell'_k$ *such that* $f(\ell'_1, \ldots, \ell'_k) \in \gamma(lab^{\theta^\star}(T^\ell)) \wedge \bigwedge_{i=1}^j \mathcal{E}_\gamma^{\theta^\star}(\ell'_i, T_i^{\ell_i})$

To prove this intermediate goal, we do structural induction on T^ℓ. Because of (e) there are only two cases to consider:

Case $T^\ell = x$. Then $\lfloor x^\ell \rfloor \theta = \theta(x) = f(U_1, \ldots, U_k)$ because of (f). Thus by definition of θ there exists ℓ such that $\theta^\star(x) = \ell$ and $f(U_1, \ldots, U_k) \in L(\gamma, \ell)$. Hence, by Definition 4.16, there exist $\ell'_1, \ldots, \ell'_k$ such that $f(\ell'_1, \ldots, \ell'_k) \in \gamma(\ell)$ and $U_i \in L(\gamma, \ell'_i)$ for $i = 1, \ldots, k$. With (f) we know $\lfloor T_i^{\ell_i} \rfloor \theta \in L(\gamma, \ell'_i)$ for $i = 1, \ldots, j$ and (b) is $\bigwedge_{i=1}^j \gamma \vDash^{\theta^\star} T_i^{\ell_i}$. Thus we can apply j times Lemma 4.19 to yield $\bigwedge_{i=1}^j \mathcal{E}_\gamma^{\theta^\star}(\ell'_i, T_i^{\ell_i})$ which concludes the case.

Case $T^\ell = f^\ell(T_1'^{\ell''_1}, \ldots, T_k'^{\ell''_k})$. It is $\lfloor f^\ell(T_1'^{\ell''_1}, \ldots, T_k'^{\ell''_k}) \rfloor \theta = f(\lfloor T_1'^{\ell''_1} \rfloor \theta, \ldots, \lfloor T_k'^{\ell''_k} \rfloor \theta) = f(U_1, \ldots, U_k)$ from (e) and therefore $\lfloor T_i'^{\ell''_i} \rfloor \theta = U_i$ for $i = 1, \ldots, k$. With (f) also $\lfloor T_i'^{\ell''_i} \rfloor \theta = \lfloor T_i^{\ell_i} \rfloor \theta$ $(*)$, for $i = 1, \ldots, j$. Since $\gamma \vDash^{\theta^\star} T^\ell$, there exist $\ell'_1 = lab^{\theta^\star}(T_1'^{\ell''_1}), \ldots, \ell'_k = lab^{\theta^\star}(T_k'^{\ell''_k})$ such that $\bigwedge_{i=1}^k \gamma \vDash^{\theta^\star} T_i'^{\ell''_i}$ and $f(\ell'_1, \ldots, \ell'_k) \in \gamma(\ell)$. We can apply Lemma 4.18 k times to have $\lfloor T_i'^{\ell''_i} \rfloor \theta \in L(\gamma, lab^{\theta^\star}(T_i'^{\ell''_i}))$ for $i = 1, \ldots, k$. With $(*)$ and the fact $\ell'_i = lab^{\theta^\star}(T_i'^{\ell''_i})$, we have also $\lfloor T_i^{\ell_i} \rfloor \theta \in L(\gamma, \ell'_i)$ for $i = 1, \ldots, j$. Using (b), we can apply j times Lemma 4.19 to have $\bigwedge_{i=1}^j \mathcal{E}_\gamma^{\theta^\star}(\ell'_i, T_i^{\ell_i})$ which concludes the case.

This concludes the proof of the intermediate goal. We can apply this goal directly to (c) to have $(\kappa^\star, \sigma^\star, \gamma) \vDash^{\theta^\star[x_i \mapsto \ell'_i]_{i=j+1}^k}_{\widehat{G},n} P_1$. We can apply the induction hypothesis since $\theta[U_i/x_i]_{i=j+1}^k(x) \in \{U : \theta^\star[x_i \mapsto \ell'_i]_{i=j+1}^k(x) = \ell'_i \wedge U \in L(\gamma, \ell'_i)\}$ for $x = x_1, \ldots, x_k$, where $U_i \in L(\gamma, \ell'_i)$ for $i = j+1, \ldots, k$ follows from the proof of the intermediate goal. Thus we have $(\kappa, \sigma) \vDash^{\theta[U_i/x_i]_{i=j+1}^k}_{\mathcal{G}(\mathcal{T}),n} P_1$ as desired.

Ad (ii). By application of the induction hypothesis.

Case $P_1 \mid P_2$. By two straightforward applications of the induction hypothesis.

Case $!P$. By straightforward application of the induction hypothesis.

Ad (2). By structural induction on labelled networks.

Case $n[P, S]$. By FIN-NODE we have $(\kappa^\star, \sigma^\star, \gamma) \vDash^{[]}_{\widehat{G},n} P$ and can immediately apply the induction hypothesis to have $(\kappa, \sigma) \vDash^{[]}_{\mathcal{G}(\mathcal{T}),n} P$. By CFA-NODE it remains to show that $\forall \lfloor U^\ell \rfloor \in \lfloor S \rfloor. (\lfloor U^\ell \rfloor, n) \in \sigma$. Fix thus $\lfloor U^\ell \rfloor \in \lfloor S \rfloor$. From FIN-NODE we know $\gamma \vDash^{[]} U^\ell$ and $(lab^{[]}(U^\ell), n) \in \sigma^\star$. We can apply Lemma 4.20 (2) to have $(\lfloor U^\ell \rfloor, n) \in \sigma$.

Case $N_1 \parallel N_2$. By two straightforward applications of the induction hypothesis. □

Appendix B

Protocol Models

B.1 μSAODV

B.1.1 μSAODV_RD_DS

```
1  signature RREQ(τ_n, τ_n, τ_n, τ_sign1) : τ_rreq message
2  signature RREP(τ_n, τ_n, τ_n, τ_n, τ_sign2) : τ_rrep message
3  signature PubKey(τ_n) : τ_pubkey secret
4  signature PrivKey(τ_n) : τ_privkey secret
5  signature Sign1(τ_privkey, τ_n, τ_n) : τ_sign1 oneway
6  signature Sign2(τ_privkey, τ_n, τ_n) : τ_sign2 oneway
7  signature Rte(τ_n, τ_n) : τ_rte secret
8  signature Keys(τ_n, τ_pubkey) : τ_rte secret
9
10 process SendRREQ(n, dst) ≝
11     out RREQ(dst, n, n, Sign1(PrivKey(n), dst, n))
12
13 process ReceiveRREQ(n) ≝
14     in RREQ(; dst, src, snd, sig1).
15     read Keys(src; pubkey)
16         checksig1 sig1 pubkey [dst, src]
17             store Rte(src, snd).
18             if dst = n then
19                 out RREP(n, src, snd, n, Sign2(PrivKey(n), n, src))
20             else
21                 out RREQ(dst, src, n, sig1)
22         else nil
23     else nil
24
25 process ReceiveRREP(n) ≝
26     in RREP(; dst, src, addr, snd, sig2).
27     if addr = n then
28         read Keys(dst; pubkey)
29             checksig2 sig2 pubkey [dst, src]
```

```
30            store Rte(dst, snd).
31            if src = n then
32                nil
33            else
34                read Rte(src; next)
35                    out RREP(dst, src, next, n, sig_2)
36                else nil
37            else nil
38        else nil
39    else nil
40
41 process P(n, dst_1, dst_2) ≝
42    SendRREQ(n, dst_1) | SendRREQ(n, dst_2) | ReceiveRREQ(n) | ReceiveRREP(n)
43
44 network N(n, dst_1, dst_2) ≝
45    n[P(n, dst_1, dst_2), Keys(n, PubKey(n)) ∪ Keys(dst_1, PubKey(dst_1))∪
46          Keys(dst_2, PubKey(dst_2))]
47
48 network System ≝ N(n_1, n_2, n_3) ‖ N(n_2, n_1, n_3) ‖ N(n_3, n_1, n_2)
49
50 topology
51    {n_1, n_2}
52    {n_2, n_3}
53    {n_1, ATT1}
54    {n_3, ATT1}
55
56 attacker 1 [n_A]
```

B.1.2 μSAODV_RD_HC

```
1  signature RREQ(τ_n, τ_int, τ_int, τ_hash, τ_sign, τ_hash) : τ_rreq message
2  signature Zero() : τ_int twoway
3  signature Succ(τ_int) : τ_int twoway
4  signature Seed(τ_n) : τ_seed secret
5  signature Hash(τ_int, τ_seed) : τ_hash oneway
6  signature PubKey(τ_n) : τ_pubkey secret
7  signature PrivKey(τ_n) : τ_privkey secret
8  signature Sign(τ_privkey, τ_n, τ_int, τ_hash) : τ_sign oneway
9  signature Rte(τ_n, τ_int) : τ_rte secret
10 signature Keys(τ_n, τ_pubkey) : τ_rte secret
11
12 process SendRREQ(n, dst) ≝
13    out RREQ(n, #0, #2, Hash(#2, Seed(n)),
14          Sign(PrivKey(n), n, #2, Hash(#2, Seed(n))), Hash(#0, Seed(n)))
15
16 process ReceiveRREQ(n) ≝
```

```
17    in RREQ(; src, hop, maxhop, tophash, sig, hash).
18    read Keys(src; pubkey)
19        checksig sig pubkey [src, maxhop, tophash]
20            case tophash of Hash(maxhop; seed)
21                case hash of Hash(hop, seed; )
22                    store Rte(src, hop).
23                    out RREQ(src, Succ(hop), maxhop, tophash, sig,
24                            Hash(Succ(hop), seed))
25                else nil
26            else nil
27        else nil
28    else nil
29
30 network N1(n) ≝
31    n[ReceiveRREQ(n), Keys(n1, PubKey(n1))∪
32          Keys(n2, PubKey(n2)) ∪ Keys(n3, PubKey(n3))]
33
34 network N2(n, dst) ≝
35    n[SendRREQ(n, dst) | ReceiveRREQ(n), Keys(n1, PubKey(n1))∪
36          Keys(n2, PubKey(n2)) ∪ Keys(n3, PubKey(n3))]
37
38 network System ≝ N2(n1, n3) ‖ N1(n2) ‖ N1(n3)
39
40 topology
41    (n1, n2)
42    (n2, n3)
43    {n1, ATT1}
44    {n3, ATT1}
45
46 attacker 1 [nA]
```

B.1.3 μSAODV_RM

```
1  signature DATA(τn, τn, τn, τdat) : τdata message
2  signature RERR(τn, τpubkey, τn, τsign) : τrerr message
3  signature TT() : τbool twoway
4  signature FF() : τbool twoway
5  signature PubKey(τn) : τpubkey secret
6  signature PrivKey(τn) : τprivkey secret
7  signature Sign(τprivkey, τn, τpubkey) : τsign oneway
8  signature Rte(τbool, τn, τn) : τrte secret
9  signature Dat() : τdat twoway
10
11 process SendDATA(n, dst) ≝
12    read Rte(TT(), dst; next)
13        out DATA(dst, next, n, Dat())
```

```
14     else nil
15
16 process ReceiveDATA(n) ≝
17     in DATA(; dst, addr, snd, data).
18     if addr = n then
19         if dst = n then
20             store data
21         else
22             read Rte(TT(), dst; next)
23                 out DATA(dst, next, n, data)
24             else
25                 read Rte(FF(), dst; next)
26                     SendRERR(n, dst)
27                 else nil
28     else nil
29
30 process SendRERR(n, unr) ≝
31     out RERR(unr, PubKey(n), n, Sign(PrivKey(n), unr, PubKey(n)))
32
33 process ReceiveRERR(n) ≝
34     in RERR(; unr, pubkey, snd, sig).
35     checksig sig pubkey [unr, pubkey]
36         out RERR(unr, PubKey(n), n, Sign(PrivKey(n), unr, PubKey(n))).
37         read Rte(TT(), unr, snd; )
38             store Rte(FF(), unr, snd)
39         else nil
40     else nil
41
42 process P(n) ≝ ReceiveDATA(n) | ReceiveRERR(n)
43
44 network System ≝
45     n1[P(n1) | SendDATA(n1, n4), Rte(TT(), n4, n2) ∪ Rte(TT(), n6, n2)]
46   ‖ n2[P(n2), Rte(TT(), n4, n3) ∪ Rte(TT(), n1, n1) ∪ Rte(TT(), n6, n5)]
47   ‖ n3[P(n3), Rte(TT(), n4, n4) ∪ Rte(TT(), n1, n2)]
48   ‖ n4[P(n4), Rte(TT(), n1, n3)]
49   ‖ n5[P(n5) | SendRERR(n5, n6), Rte(TT(), n1, n2)]
50
51 topology
52     {n1, n2}
53     {n2, n3}
54     {n2, n5}
55     {n3, n4}
56     {n3, ATT1}
57     {n5, ATT1}
58
59 attacker 1 [nA, PrivKey(nA), PubKey(nA)]
```

B.2 μARAN

B.2.1 μARAN_RD

```
1  signature RDP(τn, τcert1, τsign1, τsign2, τcert) : τrdp message
2  signature REP(τn, τn, τcert, τsign1, τsign2, τcert) : τrep message
3  signature Cert(τn, τpubkey, τsign3) : τcert twoway
4  signature Certs(τcert) : τcerts secret
5  signature PubKey(τn) : τpubkey secret
6  signature PrivKey(τn) : τprivkey secret
7  signature Sign1(τprivkey, τn, τcert) : τsign1 oneway
8  signature Sign2(τprivkey, τn, τcert, τsign1) : τsign2 oneway
9  signature Sign3(τprivkey, τn, τpubkey) : τsign3 oneway
10 signature Rte(τn, τn) : τroute secret
11
12 process SendRDP(dst, n) ≝
13     read Certs(; cert)
14         out RDP(dst, cert, Sign1(PrivKey(n), dst, cert), Sign2(PrivKey(n), dst, cert,
15                 Sign1(PrivKey(n), dst, cert)), cert)
16     else nil
17
18 process ReceiveRDP(n) ≝
19     in RDP(; dst, cert1, sig1, sig2, cert2).
20     case cert1 of Cert(; src, pubkey1, sigcert1)
21         checksig3 sigcert1 PubKey(srv) [src, pubkey1]
22             checksig1 sig1 pubkey1 [dst, cert1]
23                 case cert2 of Cert(; snd, pubkey2, sigcert2)
24                     checksig3 sigcert2 PubKey(srv) [snd, pubkey2]
25                         checksig2 sig2 pubkey2 [dst, cert1, sig1]
26                             store Rte(src, snd).
27                             read Certs(; cert)
28                                 if dst = n then
29                                     out REP(snd, src, cert, Sign1(PrivKey(n), src, cert),
30                                             Sign2(PrivKey(n), src, cert,
31                                                 Sign1(PrivKey(n), src, cert)), cert)
32                                 else
33                                     out RDP(dst, cert1, sig1,
34                                             Sign2(PrivKey(n), dst, cert1, sig1), cert)
35                             else nil
36                         else nil
37                     else nil
38                 else nil
39             else nil
40         else nil
41     else nil
42
43 process ReceiveREP(n) ≝
```

```
44    in REP(n; src, cert_1, sig_1, sig_2, cert_2).
45    case cert_1 of Cert(; dst, pubkey_1, sigcert_1)
46        checksig_3 sigcert_1 PubKey(srv) [dst, pubkey_1]
47            checksig_1 sig_1 pubkey_1 [src, cert_1]
48                case cert_2 of Cert(; snd, pubkey_2, sigcert_2)
49                    checksig_3 sigcert_2 PubKey(srv) [snd, pubkey_2]
50                        checksig_2 sig_2 pubkey_2 [src, cert_1, sig_1]
51                            store Rte(dst, snd).
52                            read Certs(; cert)
53                                if src = n then
54                                    nil
55                                else
56                                    read Rte(src; next)
57                                        out REP(next, src, cert_1, sig_1,
58                                                Sign_2(PrivKey(n), src, cert_1, sig_1), cert)
59                                    else nil
60                            else nil
61                        else nil
62                    else nil
63                else nil
64            else nil
65        else nil
66    else nil
67
68 network System ≝
69        n_1[SendRDP(n_3, n_1) | ReceiveREP(n_1), Certs(C_{n_1})]
70    ‖ n_2[ReceiveRDP(n_2) | ReceiveREP(n_2), Certs(C_{n_2})]
71    ‖ n_3[ReceiveRDP(n_3), Certs(C_{n_3})]
72
73 topology
74    {n_1, n_2}
75    {n_2, n_3}
76    {n_1, ATT1}
77    {n_3, ATT1}
78
79 attacker 1 [n_A]
```

B.2.2 μARAN_RM

```
1 signature DATA(τ_n, τ_n, τ_n, τ_dat) : τ_data message
2 signature ERR(τ_n, τ_n, τ_n, τ_cert, τ_sign_1) : τ_err message
3 signature Cert(τ_n, τ_pubkey, τ_sign_3) : τ_cert secret
4 signature Certs(τ_cert) : τ_certs secret
5 signature TT() : τ_bool secret
6 signature FF() : τ_bool secret
7 signature PubKey(τ_n) : τ_pubkey secret
```

```
8  signature PrivKey(τ_n) : τ_privkey secret
9  signature Sign1(τ_privkey, τ_n, τ_n, τ_cert) : τ_sign1 oneway
10 signature Sign3(τ_privkey, τ_n, τ_pubkey) : τ_sign3 oneway
11 signature Rte(τ_bool, τ_n, τ_n) : τ_rte secret
12 signature Dat() : τ_dat twoway
13
14 process SendDATA(n, dst) ≝
15     read Rte(TT(), dst; next)
16         out DATA(next, n, dst, Dat())
17     else nil
18
19 process ReceiveDATA(n) ≝
20     in DATA(n; src, dst, data).
21     if dst = n then
22         store data
23     else
24         read Rte(TT(), dst; next)
25             out DATA(next, src, dst, data)
26         else
27             read Rte(FF(), dst; next1)
28                 read Rte(TT(), src; next2)
29                     read Certs(; cert)
30                         out ERR(next2, src, dst, cert, Sign1(PrivKey(n), src, dst, cert))
31                     else nil
32                 else nil
33             else nil
34
35 process ReceiveERR(n) ≝
36     in ERR(n; src, dst, cert, sig).
37     case cert of Cert(; snd, pubkey, sigcert)
38         checksig3 sigcert PubKey(srv) [snd, pubkey]
39             checksig1 sig pubkey [src, dst, cert]
40                 read Rte(TT(), dst, snd; )
41                     store Rte(FF(), dst, snd).
42                     read Rte(TT(), src; next)
43                         read Certs(; owncert)
44                             out ERR(next, src, dst, owncert,
45                                     Sign1(PrivKey(n), src, dst, owncert))
46                         else nil
47                     else nil
48                 else nil
49             else nil
50         else nil
51     else nil
52
53 process P(n) ≝ ReceiveDATA(n) | ReceiveERR(n)
54
55 network μARAN_RM ≝
```

```
56      n1[P(n1) | SendDATA(n1, n3), Rte(TT(), n3, n2) ∪ Certs(C_n1)]
57    || n2[P(n2), Rte(TT(), n3, n3) ∪ Rte(TT(), n1, n1) ∪ Certs(C_n2)]
58    || n3[P(n3), Rte(TT(), n1, n2) ∪ Certs(C_n3)]
59
60 topology
61      {n1, n2}
62      {n2, n3}
63      {n2, ATT1}
64
65 attacker 1 [n_A, PrivKey(n_A), PubKey(n_A)]
```

Bibliography

[1] Martín Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[2] Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'02)*, pages 33–44. ACM, 2002.

[3] 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 (POPL'01)*, pages 104–115. ACM, 2001.

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

[5] Gergely Ács, Levente Buttyán, and István Vajda. Provable security of on-demand distance vector routing in wireless ad hoc networks. In *Proceedings of the 2nd European Workshop on Security and Privacy in Ad Hoc and Sensor Networks (ESAS'05)*, volume 3813 of *Lecture Notes in Computer Science*, pages 113–127. Springer, 2005.

[6] Alexander Aiken. Introduction to set constraint-based program analysis. *Science of Computer Programming*, 35(2):79–111, 1999.

[7] Alexander Aiken, Manuel Fähndrich, Jeffrey S. Foster, and Zhendong Su. A toolkit for constructing type- and constraint-based program analyses. In *Proceedings of the 2nd International Workshop on Types in Compilation (TIC'98)*, volume 1473 of *Lecture Notes in Computer Science*, pages 78–96. Springer, 1998.

[8] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[9] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[10] David Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[11] Donald Beaver. Foundations of secure interactive computing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'91)*, volume 576 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 1992.

[12] Jan A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[13] Gerard Berry and Gerard Boudol. The chemical abstract machine. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'90)*, pages 81–94. ACM, 1990.

[14] L. Bettini, V. Bono, R. De Nicola, G. Ferrari, D. Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, and B. Venneri. The Klaim project: theory and practice. In *Proceedings of the IST/FET International Workshop on Global Computing: Programming Environments, Languages, Security and Analysis of Systems (GC'03)*, volume 2874 of *Lecture Notes in Computer Science*. Springer, 2003.

[15] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. A semantics for web services authentication. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'04)*, pages 198–209. ACM, 2004.

[16] Karthikeyan Bhargavan, Carl A. Gunter, Moonjoo Kim, Insup Lee, Davor Obradovic, Oleg Sokolsky, and Mahesh Viswanathan. Verisim: Formal analysis of network simulations. *SIGSOFT Software Engineering Notes*, 25(5):2–13, 2000.

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

[18] Steve Bishop, Matthew Fairbairn, Michael Norrish, Peter Sewell, Michael Smith, and Keith Wansbrough. Engineering with logic: HOL specification and symbolic-evaluation testing for TCP implementations. In *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'06)*, pages 55–66. ACM, 2006.

[19] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Computer Society, 2001.

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

[21] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Control flow analysis can find new flaws too. In *Proceedings of the 2004 ACM SIGPLAN and IFIP WG 1.7 Workshop on Issues in the Theory of Security (WITS'04)*. 2004.

[22] C. Bodei, M. Buchholtz, P. Degano, H. Riis Nielson, and F. Nielson. Static validation of security protocols. *Journal of Computer Security*, 13(3):347–390. 2005.

[23] Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Control flow analysis for the pi-calculus. In *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 1998.

[24] Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Static analysis for secrecy and non-interference in networks of processes. In *Proceedings of the 6th International Conference on Parallel Computing Technologies (PaCT'01)*, volume 2127 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2001.

[25] Stephen D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[26] Mikael Buchholtz. *Automated Analysis of Security in Networking Systems*. PhD thesis, Technical University of Denmark, 2005.

[27] Mikael Buchholtz, Hanne Riis Nielson, and Flemming Nielson. A calculus for control flow analysis of security protocols. *International Journal of Information Security*, 2(3-4):145–167, 2004.

[28] Michael Burrows, Martín Abadi, and Roger M. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[29] Levente Buttyán and István Vajda. Towards provable security for ad hoc routing protocols. In *Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'04)*, pages 94–105. ACM. 2004.

[30] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[31] Ilaria Castellani. Process algebras with localities. In J. A. Bergstra, A. Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 945–1045. Elsevier, 2001.

[32] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, 1989.

[33] Iliano Cervesato. The Dolev-Yao intruder is the most powerful attacker. In *Proceedings of the 16th Annual Symposium on Logic in Computer Science (LICS'01)*. IEEE Computer Society, 2001. Short paper.

[34] Sibusisiwe Chiyangwa and Marta Kwiatkowska. A timing analysis of AODV. In *Proceedings of the 7th IFIP WG 6.1 International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'05)*, volume 3535 of *Lecture Notes in Computer Science*, pages 306–321. Springer, 2005.

[35] Edmund M. Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith. Verification by network decomposition. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *Lecture Notes in Computer Science*, pages 276–291. Springer, 2004.

[36] Thomas Clausen and Philippe Jacquet. Optimized link state routing protocol (OLSR). RFC 3626 (Experimental), October 2003.

[37] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available online: `http://www.grappa.univ-lille3.fr/tata`, 1997.

[38] M. Scott Corson and Joseph Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations. RFC 2501 (Informational), January 1999.

[39] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM, 1977.

[40] S. Creese, M. Goldsmith, B. Roscoe, and I. Zakiuddin. The attacker in ubiquitous computing environments: Formalising the threat model. In *Proceedings of the 1st International Workshop on Formal Aspects in Security and Trust (FAST'03)*. 2003.

[41] Satyaki Das and David L. Dill. Counter-example based predicate discovery in predicate abstraction. In *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD'02)*, volume 2517 of *Lecture Notes in Computer Science*, pages 19–32. Springer, 2002.

[42] Rocco de Nicola and Matthew C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34(1-2):83–133, 1984.

[43] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Measuring the confinement of probabilistic systems. *Theoretical Computer Science*, 340(1):3–56, 2005.

[44] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[45] N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[46] Christian Ene. *Un modèle formel pour les systèmes mobiles à diffusion.* PhD thesis, Université d'Aix-Marseille II, 2001.

[47] Christian Ene and Traian Muntean. A broadcast-based calculus for communicating systems. In *Proceedings of the 6th International Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA '03)*, 2001.

[48] Cristian Ene and Traian Muntean. Expressiveness of point-to-point versus broadcast communications. In *Proceedings of the 12th International Symposium on the Fundamentals of Computation Theory (FCT'99)*, volume 1684 of *Lecture Notes in Computer Science*, pages 258–268. Springer, 1999.

[49] Paul Erdös and Alfréd Rényi. On random graphs. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[50] Shimon Even and Oded Goldreich. On the security of multi-party ping-pong protocols. Technical Report 285, Technion (Israel Institute of Technology), 1983.

[51] F. Javier Thayer Fabrega, Jonathan Herzog, and Josua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(1):191–230, 1999.

[52] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, Lecture Notes in Computer Science, pages 406–421. Springer, 1996.

[53] David Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[54] David Gelernter and Nicholas Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, 1992.

[55] Dieter Gollmann. *Computer Security*. John Wiley & Sons, 1999.

[56] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy (S&P'90)*, pages 234–248. IEEE Computer Society, 1990.

[57] Andrew D. Gordon and Alan Jeffrey. Authenticity by typing for security protocols. *Journal of Computer Security*, 11(4):451–519, 2004.

[58] Andrew D. Gordon and Alan Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12(3-4):435–483, 2004.

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

[60] Manel Guerrero Zapata. Simple ad hoc key management (SAKM). Internet Draft, 7 February 2006.

[61] Manel Guerrero Zapata and N. Asokan. Securing ad-hoc routing protocols. In *Proceedings of the 2002 ACM Workshop on Wireless Security (WiSe'02)*, pages 1–10. ACM, 2002.

[62] Zygmunt J. Haas. A new routing protocol for the reconfigurable wireless networks. In *Proceedings of the 6th IEEE International Conference on Universal Personal Communications (ICUPC'97)*, pages 562–566, 1997.

[63] Zygmunt J. Haas, Marc R. Pearlman, and Prince Samar. The zone routing protocol (ZRP) for ad hoc networks. Internet Draft, July 2002.

[64] Nevin Heintze and J. D. Tygar. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22(1):16–30, 1996.

[65] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1), 1985.

[66] Matthew Hennessy and Julian Rathke. Bisimulations for a calculus of broadcasting systems. In *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *Lecture Notes in Computer Science*, pages 486–500, 1995.

[67] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. In *Proceedings of the 3rd International Workshop on High-level Concurrent Languages (HLCL'98)*, volume 16 of *Electronic Notes in Theoretical Computer Science*, pages 174–188. Elsevier, 1998.

[68] Carl Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8(3):323–364, 1977.

[69] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[70] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[71] Kohei Honda, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Secure information flow as typed process behaviour. In *Proceedings of the 9th European Symposium on Programming Languages and Systems (ESOP'00)*, volume 1782 of *Lecture Notes in Computer Science*, pages 180–199. Springer, 2000.

[72] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 2001.

[73] Yih-Chun Hu, David B. Johnson, and Adrian Perrig. SEAD: Secure efficient distance vector routing in mobile wireless ad hoc networks. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)*, pages 3–13, 2002.

[74] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Proceedings of the 8th ACM International Conference on Mobile Computing and Networking (MobiCom'02)*, pages 12–23. ACM, 2002.

[75] Michael R. A. Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridge University Press, 2nd edition, 2004.

[76] Philippe Jacquet, Paul Mühlethaler, Thomas Clausen, Anis Laouiti, Amir Qayyum, and Laurent Viennot. Optimized link state routing protocol. In *Proceedings of the 2001 IEEE International Multi-topic Conference (INMIC'01)*, pages 62–68, 2001.

[77] Thomas Jensen, Florimond Ployette, and Olivier Ridoux. Iteration schemes for fixed point computation. In *Proceedings of the 4th International Workshop on Fixed Points in Computer Science (FICS'02)*, 2002.

[78] 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, 1996.

[79] David B. Johnson, David A. Maltz, and Yih-Chun Hu. The dynamic source routing protocol for mobile ad hoc networks (DSR). Internet Draft, July 2004.

[80] Neil D. Jones and Steven S. Muchnick. Flow analysis and optimization of Lisp-like structures. In *Proceedings of the 6th Annual ACM Symposium on Principles of Programming Languages (POPL'79)*, pages 244-256. ACM, 1979.

[81] Richard A. Kemmerer, Catherine Meadows, and Jonathan K. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[82] Wolfgang Kiess and Martin Mauve. A survey on real-world implementations of mobile ad-hoc networks. *Ad-Hoc Networks*, 2006. To appear.

[83] John Kodumal and Alex Aiken. Banshee: A scalable constraint-based analysis toolkit. In *Proceedings of the 12th International Static Analysis Symposion (SAS'05)*, volume 3672 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2005.

[84] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977.

[85] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[86] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of the 2nd International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[87] Denis Lugiez and Philippe Schnoebelen. The regular viewpoint on PA-processes. *Theoretical Computer Science*, 274(1-2):89–115, 2002.

[88] Nancy Lynch and Mark Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3):219–246, 1989.

[89] Wenbo Mao and Colin Boyd. Towards the formal analysis of security protocols. In *Proceedings of the 6th IEEE Computer Security Foundations Workshop (CSFW'93)*, pages 147–158. IEEE Computer Society, 1993.

[90] John McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of the 1994 IEEE Symposium on Security and Privacy (S&P'94)*, pages 79–93. IEEE Computer Society, 1994.

[91] Catherine Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[92] Jonathan K. Millen. The interrogator model. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy (S&P'95)*, pages 251–260. IEEE Computer Society, 1995.

[93] Robin Milner. A calculus of communicating systems. *Lecture Notes in Computer Science*, 92, 1980.

[94] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[95] Robin Milner. *Communicating and Mobile Systems: The pi-calculus*. Cambridge University Press, 1999.

[96] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes. Parts I and II. *Information and Computation*, 100(1):1–77, 1992.

[97] David Monniaux. Abstracting cryptographic protocols with tree automata. In *Proceedings of the 6th International Static Analysis Symposium (SAS'99)*, volume 1694 of *Lecture Notes in Computer Science*, pages 149–163. Springer, 1999.

[98] 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.

[99] 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)*, volume 142 of *Electronic Notes in Theoretical Computer Science*, pages 195–213, 2006.

[100] Sebastian Nanz and Chris Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 2006. To appear.

[101] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[102] 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.

[103] Flemming Nielson and Hanne Riis Nielson. Flow logic and operational semantics. In *Proceedings of the 2nd Workshop on Higher-Order Operational Techniques in Semantics (HOOTS'97)*, volume 10 of *Electronic Notes in Theoretical Computer Science*, pages 150–169. Elsevier, 1997.

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

[105] Flemming Nielson, Hanne Riis Nielson, and René Rydhorf Hansen. Validating firewalls using flow logics. *Theoretical Computer Science*, 283(2):381–418, 2002.

[106] Flemming Nielson, Hanne Riis Nielson, and Helmut Seidl. Cryptographic analysis in cubic time. In *Proceedings of the Workshop on the Theory of Concurrency, Higher Order Languages and Types (TOSCA'01)*, volume 62 of *Electronic Notes in Theoretical Computer Science*, pages 7–23. Elsevier, 2002.

[107] Flemming Nielson, Hanne Riis Nielson, and Helmut Seidl. A succinct solver for ALFP. *Nordic Journal of Computing*, 9(4):335–372, 2002.

[108] Hanne Riis Nielson and Flemming Nielson. Flow logic: A multi-paradigmatic approach to static analysis. *The Essence of Computation: Complexity, Analysis, Transformation*, 2566:223–244, 2002.

[109] Hanne Riis Nielson, Flemming Nielson, and Henrik Pilegaard. Spatial analysis of bioambients. In *Proceedings of the 11th International Static Analysis Symposium (SAS'04)*, volume 3148 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2004.

[110] Davor Obradovic. *Formal Analysis of Routing Protocols*. PhD thesis, University of Pennsylvania, 2002.

[111] Karol Ostrovsky, K. V. S. Prasad, and Walid Taha. Towards a primitive higher order calculus of broadcasting systems. In *Proceedings of the 4th international ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'02)*, pages 2–13. ACM, 2002.

[112] Panagiotis Papadimitratos and Zygmunt J. Haas. Secure routing for mobile ad hoc networks. In *Proceedings of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS'02)*, pages 27–31, 2002.

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

[114] Charles E. Perkins, editor. *Ad Hoc Networking*. Addison-Wesley, 2001.

[115] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, July 2003.

[116] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM'94)*, pages 234–244. ACM, 1994.

[117] 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.

[118] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, University of Bonn, 1962.

[119] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[120] K. V. S. Prasad. A calculus of broadcasting systems. In *Proceedings of the 16th Colloquium on Trees in Algebra and Programming (CAAP'91)*, volume 493 of *Lecture Notes in Computer Science*, pages 338–358. Springer, 1991.

[121] K. V. S. Prasad. A calculus of value broadcasts. In *Proceedings of the 5th International Conference on Parallel Architectures and Languages Europe (PARLE'93)*, volume 694 of *Lecture Notes in Computer Science*, pages 391–402, 1993.

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

[123] K. V. S. Prasad. A prospectus for a calculus of mobile broadcasting systems. In Luca Aceto and Andrew D. Gordon, editors, *Short Contributions from the Workshop on Algebraic Process Calculi: The First Twenty Five Years and Beyond*, number NS-05-3. BRICS, 2005.

[124] Peter Y. A. Ryan and Steve A. Schneider. *The Modelling and Analysis of Security Protocols: The CSP Approach.* Addison-Wesley, 2001.

[125] Davide Sangiorgi and David Walker. *The pi-calculus: A Theory of Mobile Processes.* Cambridge University Press, 2001.

[126] Kimaya Sanzgiri, Bridget Dahill, Brian Neil Levine, Clay Shields, and Elizabeth M. Belding-Royer. A secure routing protocol for ad hoc networks. In *Proceedings of the 10th IEEE International Conference on Network Protocols (ICNP'02)*, pages 78–89. IEEE Computer Society, 2002.

[127] Peter Sewell. Global/local subtyping and capability inference for a distributed pi-calculus. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 695–706. Springer, 1998.

[128] Olin Shivers. Control flow analysis in scheme. In *Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation (PLDI'88)*, pages 164–174. ACM, 1988.

[129] Dawn Xiaodong Song, Sergey Berezin, and Adrian Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1-2):47–74, 2001.

[130] Paul F. Syverson and Paul C. Van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 1994 IEEE Symposium on Security and Privacy (S&P'94)*, pages 14–28. IEEE Computer Society, 1994.

[131] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 2003.

[132] C.-K. Toh. *Ad Hoc Mobile Wireless Networks: Protocols and Systems*. Prentice Hall, 2002.

[133] Oskar Wibling, Joachim Parrow, and Arnold Pears. Automatized verification of ad hoc routing protocols. In *Proceedings of the 24th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'04)*, volume 3235 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2004.

[134] Glynn Winskel. Event structure semantics for CCS and related languages. In *Proceedings of the 9th Colloquium on Automata, Languages and Programming (ICALP'82)*, pages 561–576. Springer, 1982.

[135] Pawel T. Wojciechowski and Peter Sewell. Nomadic Pict: Language and infrastructure design for mobile agents. *IEEE Concurrency*, 8(2):42–52, 2000.

[136] Chaoyue Xiong, Tadao Murata, and Jason Leigh. An approach to verifying routing protocols in mobile ad hoc networks using petri nets. In *Proceedings of IEEE 6th CAS Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication (MWC'04)*, pages 537–540, 2004.

[137] Chaoyue Xiong, Tadao Murata, and Jeffery Tsai. Modeling and simulation of routing protocol for mobile ad hoc networks using colored petri nets. In *Proceedings of the Conference on Application and Theory of Petri Nets (CRPITS'02)*, pages 145–153. Australian Computer Society, 2002.

[138] Hao Yang, Haiyun Luo, Fan Ye, Songwu Lu, and Lixia Zhang. Security in mobile ad hoc networks: Challenges and solutions. *IEEE Wireless Communications*, 11(1):38–47, 2004.

[139] Shahan Yang and John S. Baras. Modeling vulnerabilities of ad hoc routing protocols. In *Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'03)*, pages 12–20. ACM, 2003.

[140] Irfan Zakiuddin, Michael Goldsmith, Paul Whittaker, and Paul Gardiner. A methodology for model-checking ad-hoc networks. In *Proceedings of the 10th International SPIN Workshop on Model Checking Software (SPIN'03)*, volume 2648 of *Lecture Notes in Computer Science*, pages 181–196. Springer, 2003.