Using Process Algebra to Design Better Protocols

Peter Höfner

Abstract Protocol design, development and standardisation still follow the lines of *rough consensus and running code*. This approach yields fast and impressive results in a sense that protocols are actually implemented and shipped, but comes at a price: protocol specifications, which are mainly written in natural languages without presenting a formal specification, are (excessively) long, ambiguous, underspecified and erroneous. These shortcomings are neither new nor surprising, and well documented. It is the purpose of this paper to provide further evidence that formal methods in general and process algebras in particular can overcome these problems. They provide powerful tools that help to analyse and evaluate protocols, already during the design phase. To illustrate this claim, I report how a combination of pen-and-paper analysis, model-checking and interactive theorem proving has helped to perform a formal analysis of the Ad hoc On-Demand Vector (AODV) routing protocol.

1 The Need for Better Protocols

> "Despite the maturity of formal description languages and formal methods for analyzing them, the description of real protocols is still overwhelmingly informal. The consequences of informal protocol description drag down industrial productivity and impede research progress" Pamela Zave [45]

In computing, protocols are omnipresent: examples reach from *internet communication protocols*, such as the Simple Mail Transfer Protocol (SMTP) [33, 22] and the Transmission Control Protocol (TCP) [34], via *cryptographic protocols*, such Kerberos [27] and the MD5 Message-Digest Algorithm [37], and *routing protocols*, such the Border Gateway Protocol (BGP) [36] and the Ad hoc On-Demand Vec-

Peter Höfner (Corresponding Author)
NICTA and UNSW, Locked Bag 6016, UNSW, Sydney, NSW 1466, Australia
e-mail: peter.hoefner@nicta.com.au

tor (AODV) protocol [31], to *multimedia protocols*, such as the Session Initiation Protocol (SIP) [38].

Due to this omnipresence, protocols should satisfy a few important properties: (a) Specifications should be given in a way that they are easy to understand and implement. (b) Specifications have to support cross-vendor interaction, meaning if the same protocol is implemented by different vendors, these implementations should be compatible—different implementations of the same standard should be able to cooperate. (c) Lastly, protocols should, of course, be correct.

Many of the protocols standardised today, however, fail to satisfy at least one of these properties. This is mainly because of the state of the art in protocol design and development.

Protocols are Broken

There is a stunning number of protocols that have been standardised, but do not work as expected.

For example, Miskovic and Knightly showed that many routing protocols based on the IEEE 802.11s standard [19], proprietary protocols such as those developed by Motorola[1], Cisco[2] and others, as well as research routing protocols such as AODV-ST [35] and HOVER [25] are likely to establish non-optimal routes. This leads not only to an overhead in network traffic, but also to significant delays in packet delivery [26].

In [14] van Glabbeek et al. analysed AODV and proved that this routing protocol is not a priori loop free, i.e. data packets could be send through the network without ever reaching the intended destination. They argue that loop freedom hinges on non-evident assumptions to be made when resolving ambiguities occurring in the standard.

Zave showed that some disruptions in the ring structure of the Chord protocol cannot be repaired by the Chord ring-maintenance protocol as specified in [41][3] and [42]; hence the protocol is provable incorrect. In fact she stated that no published version of Chord is correct; however, "by selecting the right pseudocode from several papers, incorporating the right hints from the text of another paper, and fixing small flaws revealed by analysis, it is possible to come up with a 'best' version that may be correct" [46].

The Border Gateway Protocol (BGP), which is designed to exchange routing and reachability information between internet service providers (ISPs), is the last protocol to be mentioned. Varadhan, Govindan and Estrin showed that this protocol does not necessarily converge, and could persistently oscillate [43]. That means that nodes can change persistently the information about routes, although the network is assumed to be static.

[1] `http://www.wi-fiplanet.com/news/article.php/3600221`
[2] `http://www.mikrotik.com/`
[3] This paper won the 2011 SIGCOMM Test-of-Time Award.

Designing Protocols: State of The Art

As shown, many protocols do not behave as expected. The question that arises is why does this happen. Isn't it possible to correctly specify a protocol and test/prove fundamental properties before implementation and deployment? This paper illustrates that this is possible. It is, however, my belief that the state of the art of designing protocols—*rough consensus and running code*—is one of the problems, if the process is implemented as described below.

"[IETF's] working groups make decisions through a 'rough consensus' process. IETF consensus does not require that all participants agree although this is, of course, preferred. In general, the dominant view of the working group shall prevail. (However, 'dominance' is not to be determined on the basis of volume or persistence, but rather a more general sense of agreement). Consensus can be determined by a show of hands, humming, or any other means on which the WG agrees (by rough consensus, of course). Note that 51% of the working group does not qualify as 'rough consensus' and 99% is better than rough. It is up to the Chair to determine if rough consensus has been reached." [6]

In practice this usually means that somebody first creates a draft of a specification in natural language, such as English. This draft often contains an excellent idea and deep insights on how to tackle a specific problem, e.g. using sequence numbers to ensure loop freedom. This draft is then discussed by the working group and changes are applied to the textual draft. As soon as rough consensus on the (natural-language) specification is reached and as soon as there are at least two 'running' implementations, the protocol is declared to be standardised. Using this approach the IETF had major successes, such as the development and the deployment of DHCP (Dynamic Host Configuration Protocol), DNS (Domain Name System) and BGP.[4]

These successes suggest that the use of natural languages for protocol descriptions without presenting a formal specification seems to be advantageous: everybody can easily read, understand and comment on the specification, and hence, the protocol is easy to implement. However, looking at contemporary protocol developments more closely, it turns out that natural languages are no proper specification languages at all. They may be easy to understand, but this comes at a price.

– *Specifications are (excessively) long*. The description of the Session Initiation Protocol (SIP) [38], for example, is 268 pages long (and is not even self-contained); the IEEE Std 802.11$^{\text{TM}}$-2012 [20] standard, which contains a set of media access control (MAC) and physical layer (PHY) specifications for wireless networks, is 2,793 pages long.
  The sheer length of these specifications makes it nearly impossible to read and understand the full specification.
– *Specifications are ambiguous and underspecified*. It is hard—maybe impossible—to write precise and unambiguous specifications using natural languages only. Ambiguities in the Ad hoc On-Demand Vector (AODV) protocol [31], for ex-

[4] A list of IETF's successes and failures can be found at `http://trac.tools.ietf.org/misc/outcomes/`.

ample, yielded 5 open-source implementations to behave in incompatible ways, although all following the standard closely [14].
– *Protocols are neither formally analysed nor verified.* The lack of an (unambiguous) formal specification makes a formal analysis impossible. Traditional approaches to analyse protocols are simulation and test-bed experiments. While these are important and valid methods for protocol evaluation, they have limitations in regards to guaranteeing basic protocol correctness properties. Experimental evaluation is resource intensive and time-consuming, and, even after a very long time of evaluation, only a finite set of scenarios can be considered—usually, no general guarantee can be given. This problem is illustrated by Miskovic's and Knightly's discovery of limitations in AODV-like protocols (see above) that have been under intense scrutiny over many years [26].

Better Protocols are Needed, Now!

These shortcomings are neither new nor surprising, and documented in several research papers, e.g. [45] or [39, Chap. 9]. I believe that many problems could be avoided if formal protocol descriptions would accompany the textual specification, already in the design phase, before rough consensus is reached. By this, different readings of the draft, or underspecification can easily be avoided. Another reason why formal methods should be used already during the design phase is that protocols are not deployed in a lab: as soon as protocols are shipped, deployed and in (regular) use, it is nearly impossible to replace them. A classic example is BGP, which is erroneous (see above), but runs at the backbone of the Internet since 1994.

It is the purpose of the remainder of this paper to provide further evidence that formal methods in general and process algebras in particular can overcome these problems. Formal methods are mathematical approaches used to formally reason about software and hardware systems. They are used from formalising systems' requirements, and specifications and designs, through programming concepts and programming languages, to implementation. They are also used to relate different formalisations: for example refinement can be used to show that an implementation 'follows' a formal specification. Formal methods are indispensable for software and protocol engineering, especially when safety, security or correctness is considered.

In the area of protocol development they provide powerful tools that help to analyse and evaluate protocols, already during the design phase. I will illustrate this by a formal analysis of AODV [31], a routing protocol currently standardised by the IETF MANET working group. I will report how a combination of pen-and-paper analysis, model-checking and interactive theorem proving has helped to carry out the analysis. This case study shows (again) that formal methods are mature enough to support protocol design from the beginning. It is my belief that the use of formal methods could have found and prevented limitations in AODV-like protocols as reported in [26].

2 Formal Specification Languages

Formal specification languages and analysis techniques are now able to capture the full syntax and semantics of reasonably rich protocols. They are an indispensable augmentation to natural language, both for specification and analysis.

Even when formal analysis is not the final aim, the use of formal languages is useful: they are unambiguous, reduce significantly the number of misunderstandings, and clarify the overall structure. By this, they almost always avoid underspecification. Obviously, formal specification languages cannot prevent errors a priori, but they will make them less likely, and since they are unambiguous they do not allow different readings of a draft or a standard. If no formal analysis is required, it does not really matter which formalism is used. The choice of formal specification languages is numerous: it ranges from timed automata, which offer tool support via model checking (e.g. [8]), via the inductive approach, which offers interactive theorem proving support [30], to algebraic characterisations such as semirings (e.g. [15]) and process algebra (e.g. [10]). For our case study (see below), process algebra was chosen as specification language. It has the advantage that it is closely related to programming languages, and hence specifications are easy to understand by network researchers and software engineers as well, not only by theoreticians.

The Process Algebra AWN

The process algebra AWN (*Algebra for Wireless Networks*) [10] was initially developed for wireless networks such as AODV, and has therefore in-built support for node mobility, broadcast/multicast communication etc. However, AWN allows modelling any type of communicating concurrent processes, and can be used for a wide range of networks and protocols, e.g. [12].

The syntax of the AWN language, depicted in Table 1 and described below, is simple and reads much like a programming language, but it is implementation independent and has all the required ingredients to be able to formally reason about protocol and network properties, and to provide mathematically rigorous proofs.

AWN is a variant of standard process algebras [24, 17, 2, 3], extended with a *local broadcast* mechanism and a *conditional unicast* operator—allowing error handling in response to failed communications—and incorporating *data structures*.

In AWN, a protocol running in a (wireless) network is modelled as parallel composition of network nodes. On each node several processes may run in parallel. Network nodes communicate with their direct neighbours—those nodes that are currently in transmission range—using either broadcast, unicast, or an iterative unicast/multicast (called *groupcast*).

The basic components of *process expressions* are given in the first part of Table 1. A process name X comes with a *defining equation* $X(\mathtt{var}_1,\ldots,\mathtt{var}_n) \overset{def}{=} p$, where p is a process expression, and the $\mathtt{var}_i$ are data variables maintained by process X. A named process is similar to a procedure or a function: if it is called, data expressions exp_i are filled in for the variables $\mathtt{var}_i$. The process $p+q$ models choice: it may act either as p or as q, depending on which of the two is able to act at all. In

Table 1 Syntax of the process algebra AWN.

Basic primitives of (node-level) sequential process expressions	
$X(exp_1,\ldots,exp_n)$	process name with arguments
$p+q$	choice between processes p and q
$[\varphi]p$	conditional process
$[\![\mathtt{var} := exp]\!]p$	assignment followed by process p
$\mathbf{broadcast}(ms).p$	broadcast ms followed by p
$\mathbf{groupcast}(dests,ms).p$	iterative unicast or multicast to all destinations $dests$
$\mathbf{unicast}(dest,ms).p \blacktriangleright q$	unicast ms to $dest$; if successful proceed with p; otherwise with q
$\mathbf{send}(ms).p$	synchronously transmit ms to parallel process on same node
$\mathbf{receive}(\mathtt{msg}).p$	receive a message
$\mathbf{deliver}(d).p$	deliver d to application layer
Some advanced sequential process expressions	
$[\varphi]p+[\neg\varphi]q$	deterministic choice with test
$X(n) \overset{def}{=} [\![n := n+1]\!]X(n)$	example of a loop
Parallel process expressions	
ξ,p	process with valuation
$P \langle\!\langle Q$	parallel processes on the same node

a context where both are able to act, a non-deterministic decision is made. The expression $[\varphi]p$ is a conditional process—an if-statement—if the Boolean expression φ evaluates to `true` then the process acts like p, it deadlocks otherwise.[5]

The process algebra also features (arbitrary) data structures. An update to a variable `var` is performed using the assignment $\mathtt{var} := exp$, where exp is a data expression of the same type as `var`. The process $[\![\mathtt{var} := exp]\!]p$ acts as p, but under the constraint that the value of the variable `var` is now exp.

AWN always provides data types for node identifiers, sets of node identifiers, and messages; variables of these types are used to model the transmission of messages, and are denoted by $dest$, $dests$ and ms, respectively. The process $\mathbf{broadcast}(ms).p$ broadcasts (the data value bound to the expression) ms to all nodes in the network within transmission range of the sender, and subsequently acts as p; the process $\mathbf{groupcast}(dests,ms).p$ transmits ms to all destinations within transmission range that are also listed in the set $dests$, and proceeds as p. Both expressions **broadcast** and **groupcast** continue as p, independently whether the transmission (to some nodes) was successful. In contrast to this, $\mathbf{unicast}(dest,ms).p \blacktriangleright q$ tries to send the message ms to the sole destination $dest$; if successful it continues to act as p and otherwise as q.[6] It models an abstraction of an acknowledgment-of-receipt mechanism that is typical for unicast communication but absent in broadcast communication, and implemented in wireless standards such as IEEE 802.11. All these mechanisms model internode message sending; for intranode communication the

[5] In case φ contains free variables, values to these variables are chosen non-deterministically in a way that satisfies φ, if possible.

[6] The unicast is unsuccessful if the destination $dest$ is out of transmission range of the sender.

process $\textbf{send}(ms).p$ is used. This action can only take place if another process is able to receive the message.

The process $\textbf{receive}(\texttt{msg}).p$ is able to receive any message m; it then proceeds as p under the constraint that the variable `msg` is updated to m. The message m can stem from another node (**broadcast/groupcast/unicast**), from the same node (**send**), or from an application layer process. The latter is modelled by the process $\textbf{receive}(\texttt{newpkt}(d, dip)).p$, where `newpkt` generates a message containing data d to be sent from the application layer, and the intended destination address dip. Data is delivered to the application layer by the process $\textbf{deliver}(d).p$.

It is straightforward to model a protocol (running on one node) using these basic process expressions. I will show a snippet of the AODV protocol in the next section. Other well-known programming constructs, such as if-then-else or loops can easily built from them; two examples are given in the second block of Table 1.

Processes running on the same node, can be combined as $P \langle\langle Q$. Here, P and Q are valuated processes, meaning that they are either a process expression p built from the syntax presented above and equipped with a valuation function ξ, which specifies values of variables maintained by p, or a parallel process itself.

In the full process algebra [10], parallel processes (processes describing the behaviour of a single network node) are combined into an expression modelling the entire network, including information about the transmission ranges of all nodes. Since we concentrate on modelling aspects, these details do not matter—the presented constructs are sufficient to describe protocols on the level of nodes.

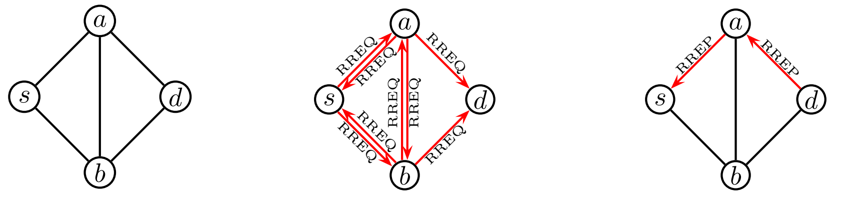
The intuition of the syntax of AWN should be clear for anybody writing protocol specifications. However, to formally reason about protocols a formal semantics is required: AWN, as many other process algebras, is equipped with an operational semantics [10]. It describes a model's behaviour in terms of its execution. As a consequence, many desired properties, such as correctness and safety can often be verified by constructing proofs from these logical statements. An example that formally describes intranode communication is given by the rule

$$\frac{P \xrightarrow{\textbf{receive}(msg)} P' \quad Q \xrightarrow{\textbf{send}(msg)} Q'}{P \langle\langle Q \xrightarrow{\tau} P' \langle\langle Q'} \ .$$

This semantical rule states that if the process Q is able to **send** a message msg and, at the same time, process P is able to **receive** the same message, then both processes will execute their actions (**send/receive**); the resulting internal action is called τ.

The main purpose of this paper is to illustrate that process algebras can be used to model and analyse reasonably rich protocols. Hence we abstain from a detailed presentation of the operational semantics.

Fig. 1 AODV by example [13].



(a) network topology (b) request floods the network (c) route reply is sent back

3 Case Study: The AODV Routing Protocol

Together with my colleagues R. van Glabbeek, M. Portmann, W.L. Tan, A. McIver and A. Fehnker, I used the process algebra AWN to obtain the first rigorous formalisation of the specification of Ad hoc On-Demand Vector (AODV) routing protocol [31]. Based on the formalisation, a careful analysis of the protocol was performed, using pen-and-paper analysis in [10, 13], the proof assistant Isabelle/HOL [28] in [5, 4], and the model checker Uppaal [1, 23] in [9, 18].

The Protocol

AODV is a reactive protocol, meaning that routes are established on demand when needed. A route from a source node s to a destination node d is a sequence of nodes $[s, n_1, \dots, n_k, d]$, where $n_1, \dots, n_k$ are intermediate nodes located on a particular path from s to d. The intuition of AODV is best illustrated by a small example, depicted in Figure 1. The network topology is given in Figure 1(a), where an edge between two nodes indicate that the nodes are within transmission range. In the example node s tries to send data packets to node d, but s does not yet have information about a route to d.

Node s initiates a route discovery mechanism by broadcasting a route request (RREQ) message, which is received by all neighbours in transmission range, nodes a and b in the example. Nodes receiving a RREQ message that do not know a route to the intended destination d re-broadcast the message. By this the RREQ message floods the network (cf. Figure 1(b)). If one of the intermediate nodes has established a route to d before, it directly sends a route reply back towards the originator s. When a node forwards a RREQ message, it updates its routing table and adds a 'reverse route' entry to s, indicating via which next hop the node s can be reached, and other information about the route.

Once the first RREQ message is received by the destination node d—we assume it stems from a—the destination node also adds a reverse route entry in its routing table, indicating that node s can be reached via node a. It then responds by sending a route reply (RREP) message addressed to node s to node a. In contrast to RREQ messages, RREP messages are unicast. Node a receives the RREP message; it creates a 'forward route' entry to d in its routing table and forwards the message to the

Fig. 2 Excerpt of AWN specification for AODV: cases for handling a RREQ message [13].[7]

```
RREQ(hops, rreqid, dip, dsn, dsk, oip, osn, sip, ip, sn, rt, rreqs, store) $\stackrel{def}{=}$
  ...
  [ dip $=$ ip ]    /* this node is the destination node */
    ...
    /* unicast a RREP towards oip of the RREQ */
    unicast(nhop(rt, oip), rrep(0, dip, sn, oip, ip)) . AODV(ip, sn, rt, rreqs, store)
    ▶ /* If the transmission is unsuccessful, a RERR message is generated */
      ...
  + [ dip $\neq$ ip ]    /* this node is not the destination node */
    (
      [dip $\in$ vD(rt) $\wedge$ dsn $\leq$ sqn(rt,dip) $\wedge$ sqnf(rt,dip) $=$ kno ]    /* valid route to dip that is fresh enough */
        ...
        /* unicast a RREP towards the oip of the RREQ */
        unicast(nhop(rt, oip), rrep(dhops(rt, dip), dip, sqn(rt, dip), oip, ip)) .
          AODV(ip, sn, rt, rreqs, store)
        ▶ /* If the transmission is unsuccessful, a RERR message is generated */
          ...
      + [ dip $\notin$ vD(rt) $\vee$ sqn(rt,dip) $<$ dsn $\vee$ sqnf(rt,dip) $=$ unk ]    /* no valid route that is fresh enough */
        /* forward RREQ message as broadcast */
        broadcast(rreq(hops $+ 1$, rreqid, dip, max(sqn(rt, dip), dsn), dsk, oip, osn, ip)) .
        AODV(ip, sn, rt, rreqs, store)
    )
```

next hop along the established reverse route. The RREP message will finally reach the originator of the RREQ message, and the route discovery process is completed and a route from *s* to *d* has been established—data packets can start to flow.

In the event of link failures, AODV uses route error (RERR) messages to inform affected nodes.

Formal Analysis

In contrast to the ambiguous de facto standard specification of AODV [31], which is written in English prose and about 35 pages long, the created AWN model is precise, yet very readable and consists only of roughly 200 lines. The model reflects precisely the intention of AODV and accurately captures all core aspects of the protocol specification, excluding all aspects of time. An excerpt, which shows the essential parts for handling a RREQ message, is given in Figure 2. The full specification as well as a detailed explanation can be found in [13]. As the semantics of AWN is completely unambiguous, specifying a protocol in such a framework enforces total precision and obviously removes any ambiguity.

An analysis of this specification revealed that under a plausible interpretation of the original specification of AODV, the protocol admits routing loops [14]; this is in direct contradiction with popular belief, the promises of the AODV standard, and the main paper on AODV [32] (with over 12,000 citations). However, we showed loop freedom of AODV under a subtly different interpretation of the original specification [13]. Our analysis, which I will report on in the remainder of this section, considered also route correctness, packet delivery, route optimality and other properties of the routing protocol. It has been carried out by pen-and-paper, with the proof assistant Isabelle/HOL [28], and with the model checker Uppaal [1, 23].

[7] As common, text placed between /* and */ are comments and not part of AWN.

- Using the formal semantics of AWN, we verified properties of AODV that can be expressed as invariants by *pen-and-paper*. Invariants are statements that hold at all times when the protocol is executed. The most important invariants were route correctness and loop freedom.
  The term *route correctness* means that all routing table entries stored at a node are entirely based on information on routes to other nodes that is currently valid or was valid at some point in the past. In case of AODV, this property is not hard to prove, but already shows the power of formal methods, since a formal proof can be provided [13].
  Loop freedom is a critical property for any routing protocol, but it is particularly relevant and challenging for wireless networks, since the underlying network topology can change constantly. Garcia-Luna-Aceves describes a loop as follows: "A routing-table loop is a path specified in the nodes' routing tables at a particular point in time that visits the same node more than once before reaching the intended destination" [11]. Packets caught in a routing loop can quickly saturate the links and can decrease the overall network performance. To the best of our knowledge we are the first to give a complete and detailed proof of loop freedom [13]. The proof of loop freedom builds on another 30 invariants that needed to be proven before loop freedom could be verified.
- Providing a pen-and-paper proof of loop freedom was a major step in the understanding of AODV, but the proof itself is about 20 pages long. To add credibility and confidence we mechanised the proof in the theorem prover Isabelle/HOL [5, 4].
  Isabelle [29] is a generic *interactive theorem prover* based on a small logical core to ease logical correctness. The main application area is the formalisation of mathematical proofs and in particular formal verification. The most widespread instance of Isabelle nowadays is Isabelle/HOL [28], which provides a higher-order logic (HOL) theorem proving environment that is ready to use for big applications. Examples for such applications are the projects *L4.verified* and *Flyspec*. L4.verified used Isabelle/HOL to prove formal functional correctness of the seL4 microkernel, a small, 3rd generation high-performance microkernel with about 8,700 lines of C code [21]. Flyspec derived a formal proof of the Kepler conjecture on dense sphere packings using the Isabelle/HOL and HOL Light proof assistants [16].
  While the hand-written process-algebraic proof of loop freedom of AODV was already very formal, the implication that transfers statements about nodes to statements about networks involves coarser reasoning over execution sequences. The mechanised proof clarifies this aspect by explicitly stating the assumptions made of other nodes. It consists of about 400 lemmas.
  Besides the added confidence that comes with having even the smallest details fastidiously checked by a machine, the real advantage in encoding model, proof, and framework in an interactive theorem prover is that they can then be analysed and manipulated (semi-)automatically.
  In [4] we showed how protocol variants, such as different readings of the textual standard or proposed improvements of the standard can quickly be analysed.

Variants often only differ in minor details, most proofs stay the same or can be adapted automatically: an interactive theorem prover tries to 'replay' the original proof and, in case of a failure, it points at all proof steps that are no longer valid.[8] One only has to concentrate on these failures. This avoids the tedious, time-consuming, and error-prone manual chore of establishing which steps remain valid for each invariant, especially for long proofs.

– *Model checking* is in particular useful to discover protocol limitations and to develop improved variants; in our setting it can be seen as a diagnostic tool that complements the other verification techniques. Model checking is limited to networks of small size—due to state space explosion—and hence cannot verify properties for all networks, in contrast to the invariant proofs mentioned above that cover all topology.
Based on our AWN-specification we developed a model of AODV for the Uppaal model checker [9]. We checked important properties, such as route correctness and route optimality, against all topologies of up to 5 nodes, which also included dynamic topologies with one link going up or down. In the case a property does not hold, Uppaal produces evidence for the fault in the form of a 'counter-example' summarising the circumstances leading to it. Such diagnostic information provides insights into the cause and correction of these failures. For some problematic and undesired behaviour of AODV, automatically found by Uppaal, we provided fixes in form of improvements of AODV, which then were (semi-)automatically verified by Isabelle/HOL.
Analysing small topologies often yields new insights, as does simulation, but the network sizes are far from realistic and quantitative information is not included. *Statistical model checking* [44, 40] can combine the systematic methodology of 'classical' model checking with the ability to analyse quantitative properties and realistic scenarios.[9] Using statistical model checking, we showed that quantitative reasoning is now feasible—for example we analysed the extent of establishing non-optimal routes—and illustrated that properties can be checked for networks of up to 100 nodes—of course an exhaustive search is not possible here.

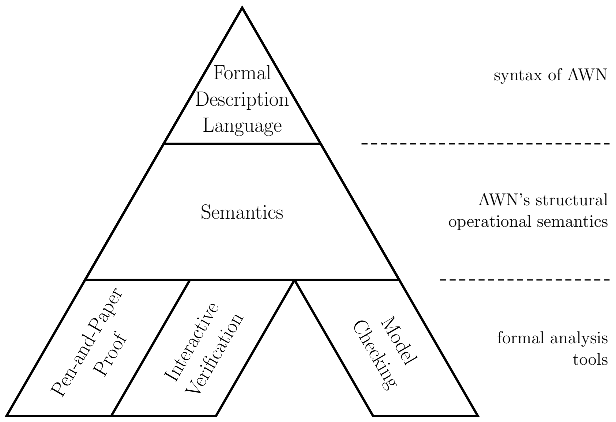
4 Looking Ahead

In this paper I have illustrated that formal specification languages and analysis techniques are now able to capture the full syntax and semantics of reasonably rich protocols. The use of formal methods in general and the use of process algebras in particular can be split into three layers (cf. Figure 3): (a) the (syntax of the) formal description language, (b) its semantics, and (c) tools for analysing a protocol, based on the syntax and the semantics. Although it would be favourable if every-

[8] [4] proves loop freedom for four variants of AODV, in average only one invariant needed major changes; and a few others needed systematic adaptions, such as changes of data types.

[9] SMC-Uppaal, the Statistical extension of Uppaal (release 4.1.11) [7] accepts the same input as standard Uppaal; the creation of a new model was not required.

Fig. 3 Different layers of Formal Methods.



body would understand all three layers, this is wishful thinking: most likely only trained experts working in the area of formal methods will understand the full spectrum. But, for specifying a protocol in a precise and unambiguous manner, which also avoids underspecification, this is not necessary. To achieve this goal, only the syntax together with a good intuition about its semantics is required—neither a full understanding of the formal semantics nor of the formal analysis tools is needed.

I believe that state-of-the-art formal description languages are simple enough to be used by any network researcher and software engineer. These languages can be used to specify and analyse rather complicated protocols. To achieve more automation in the analysis, they often offer tool support, such as model checking.

So, the question remains why despite of the maturity of formal description languages and formal methods for analysing them, the description of real protocols is still overwhelmingly informal. As Zave pointed out, this drags down industrial productivity and impedes research progress [46]. It is my belief that three ingredients are still missing: (1) *Better (easy to use) tool support*: better tools and faster computers allow more and more automation. However, the use of tools often requires special knowledge (how to use the tool) or a special input format (e.g. timed automata). (2) *Code generation:* it is often believed that the combination of formal specification followed by implementation requires more time (and hence more money) than just implementing the protocol straight away. If entire (or at least parts of) implementations could be generated out of formal specifications automatically, one could gain even more advantages from formal methods. (3) *Training:* to use formal methods, engineers working in industry must be aware of them; this can only be achieved by training. Current research tackles the first two items, the last one may be the hardest to achieve.

Acknowledgements Special thanks goes to all collaborators who contributed to the AODV case study; in particular Timothy Bourke, Ansgar Fehnker, Robert J. van Glabbeek, Annabelle McIver, Marius Portmann, and Wee Lum Tan. Further I would like to thank Robert J. van Glabbeek again for valuable comments on this paper.

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

1. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on UPPAAL. In: M. Bernardo, F. Corradini (eds.) Formal Methods for the Design of Real-Time Systems, *Lecture Notes in Computer Science*, vol. 3185, pp. 200–236. Springer (2004)
2. Bergstra, J.A., Klop, J.W.: Algebra of communicating processes. In: J.W. de Bakker, M. Hazewinkel, J.K. Lenstra (eds.) Mathematics and Computer Science, CWI Monograph 1, pp. 89–138. North-Holland (1986)
3. Bolognesi, T., Brinksma, E.: Introduction to the ISO specification language LOTOS. Computer Networks **14**, 25–59 (1987). doi:10.1016/0169-7552(87)90085-7
4. Bourke, T., van Glabbeek, R.J., Höfner, P.: A mechanized proof of loop freedom of the (untimed) AODV routing protocol. In: F. Cassez, J.F. Raskin (eds.) Automated Technology for Verification and Analysis (ATVA'14), *Lecture Notes in Computer Science*, vol. 8837, pp. 47–63. Springer (2014). doi:10.1007/978-3-319-11936-6_5
5. Bourke, T., van Glabbeek, R.J., Höfner, P.: Mechanizing a process algebra for network protocols. Journal of Automated Reasoning (2016). doi:10.1007/s10817-015-9358-9. (in press)
6. Bradner, S. (editor): IETF working group guidelines and procedures. RFC 2418 (Best Current Practice) (1998). URL `https://tools.ietf.org/html/rfc2418`
7. Bulychev, P., David, A., Larsen, K., Mikučionis, M., Bøgsted P., D., Legay, A., Wang, Z.: UPPAAL-SMC: Statistical model checking for priced timed automata. In: H. Wiklicky, M. Massink (eds.) Quantitative Aspects of Programming Languages and Systems, *EPTCS*, vol. 85, pp. 1–16. Open Publishing Association (2012)
8. Chiyangwa, S., Kwiatkowska, M.: A timing analysis of AODV. In: Formal Methods for Open Object-based Distributed Systems (FMOODS'05), *Lecture Notes in Computer Science*, vol. 3535, pp. 306–322. Springer (2005). doi:10.1007/11494881_20
9. Fehnker, A., van Glabbeek, R.J., Höfner, P., McIver, A.K., Portmann, M., Tan, W.L.: Automated analysis of AODV using UPPAAL. In: C. Flanagan, B. König (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS '12), *Lecture Notes in Computer Science*, vol. 7214, pp. 173–187. Springer (2012). doi:10.1007/978-3-642-28756-5_13
10. Fehnker, A., van Glabbeek, R.J., Höfner, P., McIver, A.K., Portmann, M., Tan, W.L.: A process algebra for wireless mesh networks. In: H. Seidl (ed.) European Symposium on Programming (ESOP '12), *Lecture Notes in Computer Science*, vol. 7211, pp. 295–315. Springer (2012). doi:10.1007/978-3-642-28869-2_15
11. Garcia-Luna-Aceves, J.J.: A unified approach to loop-free routing using distance vectors or link states. In: Symposium Proceedings on Communications, Architectures & Protocols (SIGCOMM '89), *ACM SIGCOMM Computer Communication Review*, vol. 19(4), pp. 212–223. ACM (1989). doi:10.1145/75246.75268
12. van Glabbeek, R.J., Höfner, P.: SMACCM report: Formal specification of protocols for internal high-assurance network (2015)
13. van Glabbeek, R.J., Höfner, P., Portmann, M., Tan, W.L.: Modelling and verifying the aodv routing protocol. Distributed Computing (2016). (in press)
14. van Glabbeek, R.J., Höfner, P., Tan, W.L., Portmann, M.: Sequence numbers do not guarantee loop freedom —AODV can yield routing loops—. In: Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '13), pp. 91–100. ACM (2013). doi:10.1145/2507924.2507943
15. Griffin, T.G., Sobrinho, J.: Metarouting. SIGCOMM Computer Communication Review **35**(4), 1–12 (2005). doi:10.1145/1090191.1080094

16. Hales, T.C., Adams, M., Bauer, G., Dang, D.T., Harrison, J., Le Hoang, T., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T.T., Nguyen, T.Q., Nipkow, T., Obua, S., Pleso, J., J., R., Solovyev, A., Ta, A.H.T., Tra, T.N., Trieu, D.T., Urban, J., Vu, K.K., Zumkeller, R.: A formal proof of the Kepler conjecture. CoRR (2015). URL `http://arxiv.org/abs/1501.02155`
17. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, Englewood Cliffs (1985)
18. Höfner, P., McIver, A.: Statistical model checking of wireless mesh routing protocols. In: G. Brat, N. Rungta, A. Venet (eds.) NASA Formal Methods Symposium (NFM '13), *Lecture Notes in Computer Science*, vol. 7871, pp. 322–336. Springer (2013). doi:10.1007/978-3-642-38088-4_22
19. IEEE: IEEE Standard for Information Technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 10: Mesh Networking (2011). URL `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6018236`
20. IEEE: IEEE Standard for Information Technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications (2011). (Revision of IEEE Std 802.11-2007)
21. Klein, G., Andronick, J., Elphinstone, K., Heiser, G., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: Formal verification of an operating-system kernel. Communications of the ACM **53**(6), 107–115 (2010). doi:10.1145/1743546.1743574
22. Klensin, J.: Simple mail transfer protocol. RFC 5321 (Draft Standard), Network Working Group (2008). URL `https://tools.ietf.org/html/rfc5321`
23. Larsen, K.G., Pettersson, P., Wang Yi: UPPAAL in a nutshell. International Journal of Software Tools for Technology Transfer **1**(1-2), 134–152 (1997)
24. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
25. Mir, S., Pirzada, A.A., Portmann, M.: HOVER: hybrid on-demand distance vector routing for wireless mesh networks. In: Australasian Conference on Computer Science (ACSC'08), ACSC '08, pp. 63–71. Australian Computer Society, Inc. (2008)
26. Miskovic, S., Knightly, E.W.: Routing primitives for wireless mesh networks: Design, analysis and experiments. In: Conference on Information Communications (INFOCOM '10), pp. 2793–2801. IEEE (2010). doi:10.1109/INFCOM.2010.5462111
27. Neuman, C., Yu, T., Hartman, S., Raeburn, K.: The Kerberos network authentication service (v5). RFC 4120 (Standards Track) (2005). URL `http://tools.ietf.org/html/rfc4120`
28. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, *Lecture Notes in Computer Science*, vol. 2283. Springer (2002)
29. Paulson, L.C.: The foundation of a generic theorem prover. Journal of Automated Reasoning **5**(3), 363–397 (1989). doi:10.1007/BF00248324
30. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Computer Security **6**(1-2), 85–128 (1998)
31. Perkins, C.E., Belding-Royer, E.M., Das, S.: Ad hoc on-demand distance vector (AODV) routing. RFC 3561 (Experimental), Network Working Group (2003). URL `https://tools.ietf.org/html/rfc3561`
32. Perkins, C.E., Royer, E.M.: Ad-hoc On-Demand Distance Vector Routing. In: Mobile Computing Systems and Applications (WMCSA '99), pp. 90–100. IEEE (1999). doi:10.1109/MCSA.1999.749281
33. Postel, J.B.: Simple mail transfer protocol. RFC 821 (Internet Standard) (1982). URL `https://tools.ietf.org/html/rfc821`
34. Postel, J.B. (editor): Transmission control protocol. RFC 793 (Internet Standard) (1981). URL `https://tools.ietf.org/html/rfc793`

35. Ramachandran, K., Buddhikot, M., Chandranmenon, G., Miller, S., Belding-Royer, E.M., Almeroth, K.: On the design and implementation of infrastructure mesh networks. In: IEEE Workshop on Wireless Mesh Networks (WiMesh'05)). IEEE Press (2005)
36. Rekhter, Y., Li, T., Hares, S.: A border gateway protocol 4 (BGP-4). RFC 4271 (Draft Standard), Network Working Group (Errata Exist) (2006). URL `https://tools.ietf.org/html/rfc4271`
37. Rivest, R.: The MD5 Message-Digest Algorithm. RFC 1321 (Informational, Errata Exist) (1992). URL `http://tools.ietf.org/html/rfc1321`
38. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session initiation protocol. RFC 4728 (Proposed Standard), Network Working Group (Errata Exist) (2002). URL `https://tools.ietf.org/html/rfc3261`
39. Ryan, P., Schneider, S., Goldsmith, M., Lowe, G., Roscoe, A.: The Modelling and Analysis of Security Protocols: The CSP Approach, (first published 2000) edn. Pearson Education (2010)
40. Sen, K., Viswanathan, M., Agha, G.A.: Vesta: A statistical model-checker and analyzer for probabilistic systems. In: Quantitative Evaluaiton of Systems (QEST'05), pp. 251–252. IEEE (2005)
41. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. SIGCOMM Computer Communication Review **31**(4), 149–160 (2001). doi:10.1145/964723.383071
42. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking **11**(1), 17–32 (2003). doi:10.1109/TNET.2002.808407
43. Varadhan, K., Govindan, R., Estrin, D.: Persistent route oscillations in inter-domain routing. Computer Networks **32**(1), 1–16 (2000). doi:10.1016/S1389-1286(99)00108-5
44. Younes, H.: Verification and planning for stochastic processes with asynchronous events. Ph.D. thesis, Carnegie Mellon University (2004)
45. Zave, P.: Experiences with protocol description. In: Rigorous Protocol Engineering (WRiPE' 11) (2011)
46. Zave, P.: Using lightweight modeling to understand Chord. SIGCOMM Computer Communication Review **42**(2), 49–57 (2012). doi:10.1145/2185376.2185383