Modest Models and Tools for Real Stochastic Timed Systems

Carlos E Budde, Pedro D'argenio, Juan Andrés A Fraire, Arnd Hartmanns, Zhen Zhang

Modest Models and Tools for Real Stochastic Timed Systems*

Carlos E. Budde[1], Pedro R. D'Argenio[2,3], Juan A. Fraire[4,3], Arnd Hartmanns[5] (✉), and Zhen Zhang[6]

[1] University of Trento, Trento, Italy
[2] Universidad Nacional de Córdoba, Córdoba, Argentina
[3] CONICET, Córdoba, Argentina
[4] Inria, INSA Lyon, Université de Lyon, Lyon, France
[5] University of Twente, Enschede, The Netherlands
[6] Utah State University, Logan, Utah, USA
✉ a.hartmanns@utwente.nl

Abstract. We depend on the safe, reliable, and timely operation of cyber-physical systems ranging from smart grids to avionics components. Many of them involve time-dependent behaviours and are subject to randomness. Modelling languages and verification tools thus need to support these quantitative aspects. This paper gives an introduction to quantitative verification using the Modest modelling language and the Modest Toolset. It highlights three recent case studies with increasing demands on model expressiveness and tool capabilities: A case of power supply noise in a network-on-chip modelled as a Markov chain; a case of message routing in satellite constellations that needs Markov decision processes with distributed information; and a case of optimising an attack on Bitcoin via Markov automata model checking. For each, we explain the particular conceptual and technical challenges in modelling and verification, and point out open problems for future work.

1 Introduction

Cyber-physical systems consist of discrete (usually digital, often implemented in software) controllers interacting with a continuous physical environment. Control is often networked, sometimes wirelessly. Many cyber-physical systems are safety- or performance-critical, or economically vital. We thus need to ensure that they operate as desired, which includes dependability requirements such as reliability assurances, availability levels, or response time guarantees. Reliability

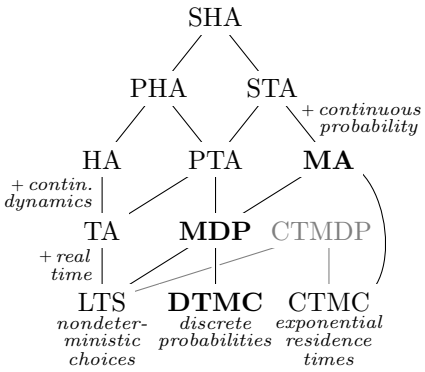
* This work was supported by Agencia I+D+i grant PICT 2022-09-00580 (CoSMoSS), the European Union's Horizon 2020 research and innovation programme under MSCA grant agreements 101008233 (MISSION) and 101067199 (ProSVED), the Interreg North Sea project STORM_SAFE, the NextGenerationEU project D53D23008400006 (SMARTITUDE) under the MUR PRIN 2022, NWO VIDI grant VI.Vidi.223.110 (TruSTy), and SeCyT-UNC grant 33620230100384CB (MECANO).

and availability are stochastic timed properties: the probability of avoiding unsafe behaviour within a certain time horizon, and the expected fraction of time that the system is ready to provide service, respectively. The critical systems themselves are also typically subject to randomisation, for example due to random message loss in wireless communication or due to employing randomised algorithms, and they are timed systems dealing with e.g. transmission delays and timeouts or faults occurring unpredictably over time. Thus, to assure their dependability by way of modelling and verification (ideally at design-time), we need stochastic timed formalisms and modelling languages supported by tools able to check stochastic timed properties.

This paper showcases the MODEST approach to modelling and verification of stochastic timed systems. MODEST, introduced by Bohnenkamp, D'Argenio, Hermanns, and Katoen in 2006 [10], was designed as a modelling language that provides process algebra-inspired modelling in a programming language-like syntax for the highly expressive model of stochastic timed automata (STA) [10], now extended to stochastic hybrid automata (SHA) [46]. Originally supported by the MoTor tool [11], today the MODEST TOOLSET [49]—in continuous development since 2008 and publicly available at modestchecker.net—provides a comprehensive collection of tools supporting the modelling, transformation, and verification of MODEST models. It notably includes the mcsta model checker [50] and the modes simulator [15]. It is part of an ecosystem of quantitative verification tools that support the interchange of models written in various modelling languages via the JSON-based JANI format [16] such as ePMC [47], Momba [61], or Storm [57]. In Sect. 2, we describe MODEST, JANI, and the MODEST TOOLSET.

MODEST and the MODEST TOOLSET have been applied to a multitude of case studies ranging from wireless ad-hoc routing protocols [59] over electricity grid stabilisation mechanisms [52] to the security evaluation of cyber-physical systems [66]. The formal modelling and analysis of any case study requires careful consideration of the modelling requirements—e.g. which kinds of quantities are relevant for the questions that the stakeholders want answered and consequently which type of underlying mathematical formalism is the most appropriate—in connection to the capabilities of the available tools. Finding the right level of abstraction is crucial as analysis techniques for more expressive types of models such as stochastic hybrid automata [36] are practically limited to much smaller model sizes compared to those for simpler models like Markov chains or decision processes [17]. Similarly, analysis techniques that provide stronger results, such as hard guarantees that the answer is within a user-specified ϵ-interval around the true optimal value, tend to be less scalable than those delivering weaker results, such as statistical guarantees or eventual convergence to the optimum only. Probabilistic model checking (PMC) [6,7], for example, is severely limited by the state space explosion problem that statistical model checking (SMC) [1] avoids entirely with its constant memory usage, at the cost of being restricted to estimation problems and delivering statistical guarantees only.

In sects. 3 to 5 of this paper, we review three different case studies that MODEST and the MODEST TOOLSET have been applied to recently. We point



Key:

SHA	stochastic hybrid automata
PHA	probabilistic hybrid automata [75]
STA	stochastic timed automata
HA	hybrid automata [2]
PTA	probabilistic timed automata [65]
MA	Markov automata
TA	timed automata [3]
MDP	Markov decision processes
CTMDP	continuous-time MDPs
LTS	labelled transition systems
DTMC	discrete-time Markov chains
CTMC	continuous-time Markov chains

Fig. 1. The family tree of automata-based quantitative formalisms

out how the specific requirements of the case study determined the choice of model type and analysis techniques, showcasing the versatility of the Modest Toolset as well as providing guidance to the modelling and verification practitioner by way of example. Our first case study (in Sect. 3) evaluates aspects of power supply noise in a two-by-two network-on-chip system by way of a discrete-time Markov chain (DTMC) model and a PMC analysis with mcsta; the second one (in Sect. 4) is about finding optimal routes through sparse constellations of nanosatellites using an abstract Markov decision process (MDP) [9,58] model analysed with an SMC-based approach that employs strategy sampling under distributed information as implemented in the modes tool; and finally (in Sect. 5) we optimise an attack on the Bitcoin cryptocurrency system via a Markov automata (MA) [33] model that permits mcsta to synthesise the strategy that minimises the expected time to success or maximises the probability of success within a certain time bound, which we turn into a human-readable decision tree representation via a new connection from mcsta to the dtControl tool [5].

This extended version. This paper is an extended version of a long presentation abstract [48] for author A. Hartmanns' invited talk at the MARS workshop at ETAPS 2022. Compared to the presentation abstract, we have expanded our introduction to Modest and Jani in Sect. 2 along concrete examples; we added more details to the presentations of the three case studies, in particular examples and insights on interesting modelling aspects for all three case studies, new research results on using Q-learning for the satellite routing case study, and results of a newly-implemented connection from mcsta to dtControl to obtain an explainable representation of the optimal attack strategy for the Bitcoin case. Throughout, we added summaries of remaining challenges and open problems.

2 Modest Languages and Tools

A well-defined semantics in terms of a mathematically well-understood object is a cornerstone of formal models. For quantitative models, we use automata-

based formalisms—that represent the evolution of a system from state to state via (randomised) transitions—building on labelled transition systems (LTS, or Kripke structures) and discrete- and continuous-time Markov chains (DTMC and CTMC, respectively) [8]. By combining these basic mathematical formalisms in various ways, and extending them with features such as real-time clocks or continuous variables evolving according to differential equations, we obtain further formalisms as depicted in Fig. 1. Since writing real-life models as, say, large Markov chains would be cumbersome, we specify them using a higher-level modelling language that offers at least discrete variables with standard arithmetic and Boolean operators plus a notion of parallel composition for the natural specification of distributed and component- or actor-based systems.

The Modest Language. One such language is MODEST, originally the modelling and description language for stochastic timed systems [10]. Its formal semantics was first defined in terms of STA [10] and later extended to SHA [46]. MODEST is a textual modelling language; its syntax is designed to be similar to widely used programming languages like C or Java to lower the barrier of entry for domain experts. At the same time, it is a process algebra in spirit, based on standard operators such as sequential and parallel composition, allowing the definition of and recursive calls to processes, and emphasising compositionality. In fact, MODEST consists of two largely orthogonal languages: one to define *behaviour*, which is the one based on process-algebraic ideas, and one to manipulate *data* such as the values of discrete variables. The latter provides arrays, recursive datatypes (e.g. allowing the definition of a linked list type via pairs of a head containing a data item and a linked list option tail), and mutually recursive functions. These features allow for concise and natural models of complex real-life systems. The properties of interest to be analysed by tools, such as queries for the maximum probability of reaching a certain goal state or requirements for the expected long-run average reward to remain below a given threshold, are specified within MODEST models as temporal logic formulas.

MODEST is equipped with a two-step semantics: The symbolic semantics maps the textual MODEST model to a network of SHA with discrete variables, where the top-level parallel composition and the values of variables are not made explicit, i.e. the composition and any operations involving variables remain as symbolic expressions. Then, the concrete semantics defines the meaning of parallel composition and variables as well as of the continuous (hybrid or timed) behaviour of the SHA, resulting in a nondeterministic labelled Markov process [30] in the most general case. As many simpler formalisms are special cases of SHA, MODEST models syntactically conforming to the restrictions of any of these sub-models have a semantics that also maps to that sub-model.

Example 1. Listing 1 shows a MODEST PTA model of a simple communication scenario: A `Sender` process transmits some file consisting of `N` data chunks over an unreliable `Channel` that loses a message with probability 0.5. One instance of the `Sender` and `Channel` processes each run in parallel, communicating by handshaking on the shared `send` and `ack` actions.

```
const int N;
action send, ack;
transient int sent;
bool done;
property ETimeDone = Xmin(T, done);
property PInTwiceN = Pmax(<>[S(sent) <= N * 2] done);
process Channel() {
   clock c;
   send palt {
   :0.5: {= c = 0 =}; when(c >= 2) invariant(c <= 4) ack // transmission success, acknowledge
   :0.5: {==}                                           // the message is lost
   }; Channel()
}
process Sender(int(0..N) i) {
   clock c;
   do {
   :: when(i > 0) send {= c = 0, sent = 1 =}; alt {
      :: ack {= i-- =}                                  // transmission succeeded
      :: when(c >= 5) invariant(c <= 5) tau             // timeout, retry chunk
      }
   :: when(i == 0) invariant(c <= 0) {= done = true =}  // done: all chunks transmitted
   }
}
par { :: Channel() :: Sender(N) }
```

Listing 1. MODEST PTA model of a communication scenario

After receiving a message to `send`, the `Channel`'s probabilistic message loss is implemented via MODEST's `palt` construct. We assume that the `Channel`'s precise transmission delay is unknown, but guaranteed to be between 2 and 4 time units because we, for example, know the message processing times and the minimum and maximum cable length or distance between wireless nodes. The timer `clock c` is used to implement this nondeterministic delay: the `ack` in `Channel` is guarded with the condition `c >= 2` and forced to execute once `c` reaches value 4 by the `invariant` condition, expressing a standard TA pattern in MODEST. The semicolon `;` is MODEST's sequential composition operator.

Nondeterministic choices between multiple behaviours are specified with the `alt` construct. It is used in the `Sender` after handing a chunk to the `Channel`: we either receive an acknowledgment, or determine that the message was lost after a timeout of exactly 5 time units. In this model, we know that, for each attempt to send, exactly one of the two possibilities will occur; in general, multiple of the choices of an `alt` can be available: a nondeterministic choice. Assignments are given in atomically-executed assignment blocks like `{= c = 0, sent = 1 =}`. Here, `sent` is a *transient* variable that is not part of the model's concrete states, but only takes values during the execution of assignment blocks.

Transient variables can also be observed by properties. In this model, we specify that we would like to know the minimum expected time until all chunks have been transmitted in property `ETimeDone`, and the maximum probability of transmitting all chunks in at most $2 \cdot$`N` attempts in property `PInTwiceN`. The former is an expected accumulated reachability reward property; the latter queries for a reward-bounded reachability probability. The way rewards are accumulated

```
{ "jani-version": 1,
  "type": "pta",
  "actions": [ { "name": "send" }, { "name": "ack" } ,
  "constants": [ { "name": "N", "type": "int" } ],
  "variables": [ { "name": "sent", "type": "int", "transient": true, "initial-value": 0 },
                 { "name": "done", "type": "bool", "initial-value": false } ],
  "properties": [ ... ],
  "automata": [
   { "name": "Channel",
     "locations": [ { "name": "loc_1" },
                    { "name": "loc_6",
                      "time-progress": { "exp": { "op": "≤", "left": "c", "right": 4 } } } ],
     ...
     "edges": [ { "location": "loc_1", "action": "send",
                  "destinations": [
                   { "location": "loc_6",
                     "probability": { "exp": { "op": "/", "left": 1, "right": 2 } },
                     "assignments": [ { "ref": "c", "value": 0 } ] },
                   ... ] },
                ... ]
   }, ... ],
  "system": { "elements": [ { "automaton": "Channel" }, { "automaton": "Sender" } ],
              "syncs": [ { "synchronise": [ "send", "send" ], "result": "send" },
                         { "synchronise": [ "ack", "ack" ], "result": "ack" } ] }
}
```

Listing 2. JANI translation of the MODEST communication PTA model (excerpt)

is specified within the properties, with `T` being a shortcut for `T(1)`—accumulate a reward of 1 per time unit—and `S`(*exp*) specifying that a reward of *exp* is accumulated every time the system transitions from one state to another.

PTA are a good fit for modelling communication protocols and networking scenarios: their real-time features capture transmission delays and timeouts, while probabilistic choices stem from the environment (such as random message loss) and the use of randomised algorithms (like exponential backoff schemes). Consequently, the AODV routing protocol was modelled in MODEST's PTA subset [59].

The JANI model interchange format. While MODEST is a convenient modelling language for end-users, implementing a MODEST parser and its symbolic semantics is a significant effort. The same problem affects many other modelling languages, e.g. Prism's [63], too. To ease tool development and facilitate the exchange of models between different tools, in 2016, the developers of several quantitative verification tools defined the JSON-based JANI [16] format. It is not designed to be human-writable, but rather serve as a model interchange format that is generated by tools from other modelling languages, such as MODEST. Today, JANI is supported by the MODEST TOOLSET (see below), ePMC [47], Storm [57], Momba [61], and several other tools. All models in the quantitative verification benchmark set (QVBS) [54] are available in both their original formats as well as in JANI. The QVBS served as the foundation for the QComp 2019 [45] and QComp 2020 [17] tool competitions.

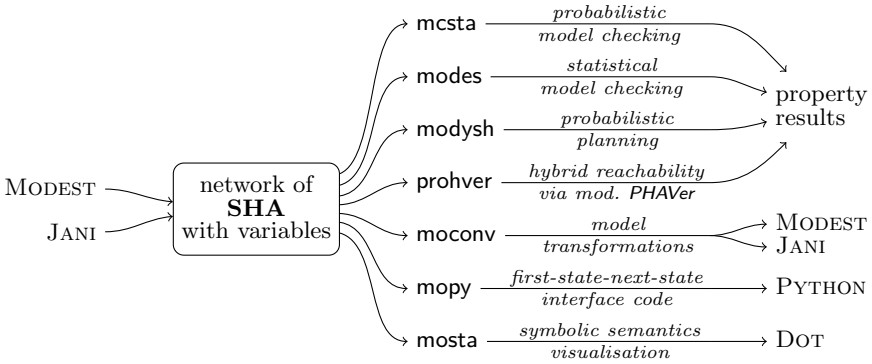


Fig. 2. Schematic overview of the Modest Toolset

Example 2. In Listing 2, we show an excerpt of the Jani translation of the Modest communication PTA model of Example 1. It is clear that Jani is not suited for human reading and quick understanding, but it is editable and inspectable. The Jani model starts with the same declarations of global items—actions, constants, global variables (discrete and continuous), and properties—as the Modest model. Then, every element of the top-level `par`allel composition in Modest is represented as one `automaton` in Jani. These automata correspond to Modest's symbolic semantics, in which Modest's textual control flow is turned into a control flow graph representation; its nodes appear as `locations` in Jani. We show some details of the `Channel` process' automaton: `invariant(c <= 4)` appears as location `loc_6`'s `time-progress` condition; every occurrence of an action in the Modest model generates an edge in the automaton, of which we show the `send`-labelled one. All expressions are represented as syntax trees in Jani, which removes much of the complications of writing a parser. The Modest model's top-level `par` determines the Jani `system` element: the parallel composition of one instance each of the `Channel` and `Sender` automata. While handshaking on shared actions is implicit in Modest, Jani uses *synchronisation vectors* (inspired by CADP's exp.open tool [38]) in the `syncs` array to explicitly describe which actions from each parallel component synchronise.

The Modest Toolset. To support the creation of Modest models, and to compute the values of properties or check requirements specified as part of models, the Modest Toolset [49] provides a collection of visualisation, model transformation, model checking, and simulation tools. The Modest Toolset has been in development since 2008; it is written in C#, and is available as precompiled binaries for common Linux distributions, macOS, and Windows running on x86-64 and ARM-64 platforms at modestchecker.net. An overview of the Modest Toolset's components is shown in Fig. 2.

As input languages, the Modest Toolset supports Modest and Jani; its moconv tool can convert between the two and apply various transformations,

such as converting a suitable PTA model into its digital clocks [64] MDP. The mosta tool visualises a model's symbolic semantics, helping in learning MODEST and in debugging models. The mopy tool converts a model into Python code implementing a first-state-next-state interface [11] that can be used to quickly prototype explicit-state verification algorithms.

The mcsta [50] tool implements PMC in an explicit-state fashion with a unique disk-based approach to mitigate the state space explosion problem. It includes efficient model reductions such as the essential states abstraction [28], and provides state-of-the-art algorithms for model checking MA [20]. A variant of mcsta implements a symblicit approach that can tackle very large models of certain structures by way of binary decision diagram (BDD)-based exploration followed by incremental explicit state elimination [44].

The statistical model checker modes [15] complements mcsta's capabilities for cases where model checking cannot be applied, such as when facing state space explosion or models with non-Markovian probability distributions like STA. SMC is, in essence, Monte Carlo simulation applied to formal models and properties. A constant-memory technique, it however incurs a runtime explosion when faced with rare events (as a prohibitively large number of samples would be needed to obtain an error that is smaller than the low probability of the rare event itself), and does not directly support nondeterministic models such as MDP. The modes tool addresses these shortcomings by providing rare event simulation [74] via a highly automated implementation of importance splitting [13], and by offering lightweight strategy sampling [67] for MDP, PTA [26,55], and (with limitations) stochastic-time models like MA and STA [27].

Finally, modysh [60] provides variants of the probabilistic planning algorithm LRTDP [12] for MDP, and prohver [46] implements an abstraction-based approach to safety verification of SHA that internally employs a modified version of the PHAVer [37] HA model checker for the hybrid reachability analysis.

Example 3. The JANI model of Example 2 was obtained by moconv from Example 1, which we model-check via

```
modest mcsta pta.modest -E "N=6"
```

to run mcsta with model parameter `N` set to 6 and using the value iteration algorithm. As a result, mcsta reports values (that are usually but not always [41] very close to the true value) of 41.99997556955714 for `ETimeDone` and 0.61279296875 for `PInTwiceN`. To obtain results with guaranteed error of at most $\pm\epsilon$ instead, we can use one of mcsta's sound value iteration algorithms: interval iteration [42], sound value iteration [71], or optimistic value iteration [53]. By adding command-line parameter `--alg IntervalIteration`, we get `ETimeDone` $= 42.00059918910847 \pm \epsilon$ and `PInTwiceN` $= 0.61279296875 \pm \epsilon$ with mcsta's default $\epsilon = 10^{-3}$. For `ETimeDone`, mcsta in fact computes the entire cumulative distribution function (CDF) as plotted in Fig. 3, i.e. the reward-bounded probability for all bounds from 0 up to the specified $12 = 2 \cdot$ `N`, via its sequential interval iteration algorithm [43].

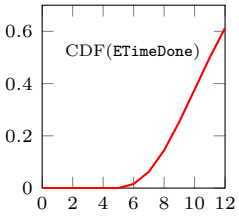


Fig. 3. CDF over `N`

If we run modes on this model with default settings, we get an error message:

`pta.modest: error: Encountered temporal nondeterminism for ack.`

This is because SMC only solves an *estimation* problem: it samples a large number of model executions (called *simulation runs*) to return the averages of the executions' property values as estimates for the expected value or probability. Yet this model poses an *optimisation* problem: find the resolutions of the nondeterministic choices that deliver the minimum expected reward or maximum probability. The error message points to the nondeterminstic choice that caused `modes` to abort: the undetermined transmission delay $\in [2, 4]$ time units. If we fix all transmission delays to 2 time units by calling `modes` with parameter `-S ASAP`, we get `ETimeDone` $= 42.00375880801009$ with a 95 % confidence interval of $[41.99375880998429, 42.01375880603589]$, and `PInTwiceN` $= 0.6092194570135746$ with the (a priori) statistical guarantee that obtaining a result within ± 0.01 of the true value of `PInTwiceN` has probability 0.95.

3 Power Supply Noise in a Network-On-Chip System

As the complexity of distributed many-core systems advances, the *network-on-chip* (NoC) architecture has become the de-facto standard for on-chip communication. A NoC is typically composed of topologically homogeneous routers operating synchronously in a decentralized manner using a predefined routing protocol. *Power supply noise* (PSN) can significantly influence the performance of the transistor devices in a NoC. PSN is created by the simultaneous switching of logic devices, which causes a drop in the effective power supply voltage. PSN is composed of two major components: resistive noise (related to the current drawn and the resistance of the circuit) and inductive noise (which is proportional to the rate of change of current through the inductance of the power grid).

To study PSN in NoC architectures, we first focused on a single central NoC router [68] and later expanded to a 2×2 NoC consisting of four symmetric routers as shown in Fig. 4. We focus on the latter in this section. Each router has three buffers responsible for storing incoming network data packets, called *flits*, and each buffer can store up to four flits. One of the buffers takes flits generated locally at the router (e.g. at a CPU attached via this router) while the other two buffer flits received from the adjacent routers (which may be destined for this router, e.g. for its attached CPU).

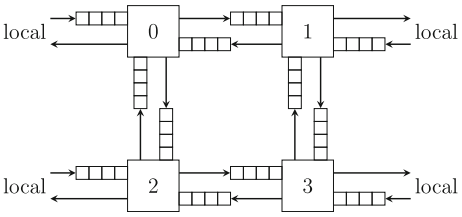


Fig. 4. Architecture of the 2×2 NoC [73]

Our goal is to compute the probability for behavioural patterns that are likely to result in resistive resp. inductive noise to occur at least n times within t clock cycles, starting from an initial state where all buffers are empty. We consider two different flit generation patterns: one where each router receives a flit into its local buffer (e.g. from the one core it is connected to) every other cycle, and one where flits are generated in bursts. We assume the destination of a flit to be one

```
// Datatype for unbounded lists of integers to represent queues of flits with destinations
datatype intlist = { int(0..NOCSIZE * NOCSIZE - 1) hd, intlist option tl };
// Datatype for a channel connecting two routers
datatype channel = { int(-1..4) direction,              // direction of buffer (N/S/W/E)
                     bool serviced, int(0..2) priority, // for round−robin protocol
                     intlist option buff };             // queue of flits to transmit
// Datatype for a router, containing incoming channels and some router−specific variables
datatype router = { channel[] channelArray, int(0..3) unserviced, ... };
// The whole NoC consists of 4 router instances with initially empty channels
router[] noc = [ router { unserviced: 0 }, router { unserviced: 0 },
                 router { unserviced: 0 }, router { unserviced: 0 } ];
```

Listing 3. User-defined datatypes in the concrete MODEST 2×2 NoC model

of the other router's local outputs, with the actual router selected uniformly at random for each flit. The routers use X-Y routing, where a flit is first routed in the horizontal direction, and a round-robin-style protocol to handle contention.

A case for DTMC model checking. With all decisions fixed to be either deterministic (flit generation times and routing choices) or random (flit destinations), and the whole NoC running on a discrete clock, this system can naturally be modelled as a DTMC. DTMC are well-suited for SMC due to the absence of nondeterminism, but there are arguments to use PMC with mcsta instead: First, a PMC analysis can deliver guaranteed ϵ-precise results (as in Example 3), whereas SMC only provides weaker statistical guarantees. Second, as we will see below, some of the probabilities we want to compute especially for small numbers of clock cycles t are very low, thus we would need to engage modes' rare event simulation engine whose automation features do not match well with the structure of the NoC model. Finally, and most importantly, we would actually like to obtain the full CDF for resistive and inductive noise events over increasing values of t. The sequential iteration approach implemented in mcsta that we already saw in Example 3 would be able to compute this CDF very efficiently, with hard guarantees on the results. The main challenge for model checking with mcsta is then to deal with the state space explosion problem.

A sequence of abstractions. We started by creating a detailed MODEST model of the 2×2 NoC, exploiting the availability of user-defined datatypes in MODEST to represent the state of the network's routers and buffers in full detail. To the best of our knowledge, MODEST and Uppaal's TA modelling interface are the only language for quantitative verification that support the definition of complex datatypes, with mcsta or modes and Uppaal or Uppaal SMC [18,32] being the only probabilistic/timed or statistical model checkers, respectively, supporting models with such datatypes. We show the declarations of the datatypes for the *concrete* 2×2 NoC model in Listing 3. A `router` mainly consists of its incoming `channel`s, which in turn contain a buffer of flits plus the information needed for the router's round-robin algorithm to determine which channel to serve next in

case of contention. A flit is simply an integer representing its destination; the buffers are this simple functional-style lists of integers of type `intlist`. The last line of Listing 3 defines the variable representing the state of the full 2×2 NoC as an array of four individual router datatype instances.

mcsta runs out of memory during state space exploration on the concrete model. Thus we cannot apply sequential iteration algorithm; however, by making the clock cycle counter a state variable (and thus effectively "unfolding" the state space over the counter's values), we were able to perform model checking for up to $t = 4$ clock cycles. This is because initially, few flits are present throughout the system, and thus the number of combinations of buffer occupancies with different flits remains small over the first few clock cycles. Nevertheless, $t = 4$ is much too low to be useful: with bursty flit generation, we have 3 flit-generating cycles every 10 clock cycles, so we do not even cover one burst cycle.

We then manually applied a series of abstractions to achieve tractability. We first applied predicate abstraction, transforming the model to replace all complex datatype instances by predicate variables that capture the model's critical decision points. For example, the two predicate variables

```
bool r0L1; // for noc[0].channel[local].direction == east
bool r0L2; // for noc[0].channel[local].direction == south
```

capture the two possible forwarding directions of the front flit in the local channel buffer of router 0. This only delayed running out of memory to $t = 7$.

The next step is a novel *probabilistic choice abstraction*: A flit's destination is uniformly randomly selected when it is *put into* the local buffer of a router, but the destination information is not checked until the flit enters the router, to decide the forwarding direction. Thus the random choice of the destination can be delayed until the flit is *taken from* the buffer. In abstract terms, probabilistic choice abstraction delays the resolution of a probabilistic choice until its evaluation point, then removes relevant state variables by replacing them with an explicit probability distribution. We implement this abstraction by modifying the MODEST code accordingly, i.e. manually and on the syntax level.

Finally, as a result of the previous abstraction steps, we can eliminate some buffer priority orders, then observe that the X-Y routing scheme now makes it unnecessary to keep track of each flit's destination, and so replace the buffer queues by bounded integer variables counting the number of waiting flits only.

In Fig. 5, we show the impact of these abstractions on the number of states (in millions) explored in the unfolded model for increasing clock cycle bound t (blue circles: concrete model, red triangles: after predicate abstraction, purple diamonds: probabilistic choice abstraction, green squares: final).

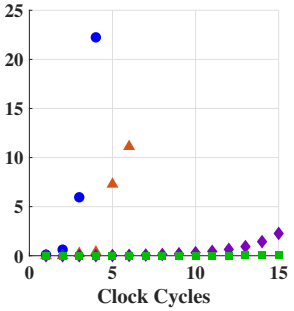


Fig. 5. State count [73]

PMC results. The final model was still too large to fully explore in non-unfolded form (i.e. treating clock cycles as rewards, like variable `sent` in Example 1) under the every-other-cycle flit generation pattern. By unfolding the clock cycle counter into the state space as described above, PMC became possible for up to 30 clock cycles. With bursty flit generation, however, we could build the non-unfolded state space and thus apply the sequential iteration technique to compute the entire CDF as shown in Fig. 6. The difference between the two patterns is that, with every-other-cycle generation, the buffers slowly fill up with flits to various destinations; the full state space that includes all combinations of buffer occupancies with different flits is too large to handle today. With bursty flit generation, however, all buffers periodically return to an empty state; the period is small enough for the entire state space to fit into memory, i.e. buffers do not fill up far enough for the number of combinations of buffer states to grow too large.

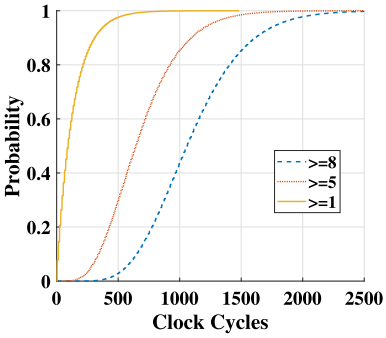


Fig. 6. Inductive noise events [73]

SMC and BDDs. We also applied SMC, which however was limited in the case of every-other-cycle generation by noise events being relatively rare, and in the case of bursty generation by not being able to compete in terms of runtime with the sequential iteration technique. Similarly, our attempts to use Storm's BDD-based state space exploration did not provide scalability improvements, possibly due to the model not being as structured as we think it is, or simply due to a bad variable ordering in the model. For further details on this first case study, we refer the reader to the original FMICS 2021 paper [73].

Open problems. The probabilistic choice abstraction was manually applied to the NoC model on the Modest code level; we would like to generalise and automate it. This case study also highlights the need to further improve the scalability of PMC; we need to investigate why the standard approach to handle large models—using BDDs—failed here and what alternatives could be.

Data availability. The Modest models described above from [73] are available at github.com/formal-verification-research/Modest-Probabilistic-Models-for-NoC.

4 Routing in Satellite Constellations

Satellite networks in low-Earth orbit are increasingly used to collect and distribute information across the globe, including access to the Internet. Real-time applications like Internet access require very large constellations (such as the SpaceX's Starlink constellation); even when using low-cost satellites based on

off-the-shelf components that are not space-qualified, such constellations are extremely expensive. A different and more sustainable approach is to relax the real-time constraint and leverage the store-carry-and-forward principle where nodes store received messages for later forwarding to other nodes in the network, once a communication window appears. The result is a *delay-tolerant network* (DTN).

In satellite constellations, the orbits are known with sufficient precision to calculate upcoming *contacts*, i.e. communication windows, over the next few days, giving rise to a *contact plan*. However, message transmissions may fail for various reasons such as unreliable (low-cost) components, contact mispredictions, or interference during the wireless communication. If statistical data is available or the error margins of calculations are known, we can assign a success probability to each contact, giving rise to an *uncertain* contact plan. Fig. 7 shows an abstract representation of such a plan. This artificial example comprises four satellites (or ground stations) N_1 to N_4 with contacts over time slots T_1 to T_5. The numbers annotating contacts are the transmission success probabilities. Given a message's source and destination, and a limit n on the number of message copies present in the network to avoid exhausting the satellites' limited resources, we would like to compute the routing strategy that maximises the probability of message delivery within the time window covered by the contact plan.

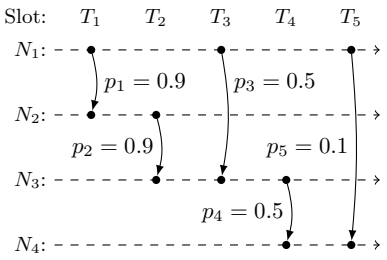


Fig. 7. Uncertain contact plan [23]

A case for MDP with distributed information. Due to the combination of randomness (in transmission failures) with nondeterministic decisions to be optimised (which contacts to use to send how many copies) in a discrete-time setting (a sequence of contacts), MDP are the perfect match among the formalisms of Fig. 1 to model this problem. The goal is to find an optimal (routing) strategy in the MDP. This looks like the perfect job for PMC, which, by computing the maximum message delivery probability, would implicitly also compute the corresponding optimal strategy. By running `mcsta` with the `--write-scheduler` parameter, it would create a text file mapping every state of the MDP to the strategy's choice among that state's enabled actions.

However, PMC works with complete, global information. Consider the contact plan of Fig. 7 with $n = 2$ copies to send: N_1 will send one copy to N_2 in slot T_1, and if successful, N_2 will forward it in slot T_2. In slot T_3, the best course of action computed by PMC for satellite N_1 is to send its remaining copy to N_3 *if and only if* N_3 did not receive the first copy. Thus the model checker, and the strategy it computes, "sees" the state of all satellites. Yet satellites do not have global information about the state of all other satellites in the constellation, making the optimal strategies found by PMC potentially unimplementable.

In fact, what we need are distributed strategies [39]. Unfortunately, the model checking problem under distributed strategies is undecidable, and even with sim-

plifications such as restricting to memoryless strategies, it remains practically intractable [40]. Recently, L-RUCoP, an approximative model checking-based approach specifically tailored to the uncertain DTNs case, has become available [72], which, however, remains limited by state space explosion as n increases.

Using SMC to find distributed strategies. We instead adapted two methods for finding (near-)optimal strategies with SMC to the distributed-information setting [24]: lightweight strategy sampling (LSS) [67], and the reinforcement learning [76] technique of Q-learning [77]. We implemented both in modes.

LSS. The key idea of LSS is to represent each strategy with a fixed-size (e.g. 32-bit) integer, i.e. in constant memory. LSS with the smart sampling heuristics [29] then randomly samples m strategies (i.e. integers), performs k simulation runs computing the average message delivery probability for each, discards the $\lceil \frac{m}{2} \rceil$ worst-performing strategies, simulates the remaining strategies with $2k$ runs each, and so on until only one "optimal" strategy remains. For this one, the message delivery probability is subsequently estimated in a standard SMC analysis. The result is a best-effort underapproximation of the maximum probability achievable with the unknown optimal strategy. During simulation for strategy i, when the simulator needs to choose between $k > 1$ actions in state s, it concatenates the bitstring representations of s and i, applies a hash function $\mathcal{H}$ mapping this value to a fixed-size integer j, and selects the $((j \mod k) + 1)$-th action:

$$a := ((\mathcal{H}(\sigma.s) \bmod |\{\text{ actions from } s\}|) + 1)\text{-th element of } \{\text{ actions from } s\}$$

To perform the same analysis w.r.t. distributed strategies, all we need to change is the input to $\mathcal{H}$: instead of the bitstring for s, we use that for a projection of s to the variables observable by the currently active component (here: satellite). We also introduce a condition of good-for-distribution models that, when satisfied, ensures that no two components may have a decision at the same time instant, making a global arbiter to break such ties unnecessary [23].

Q-learning stores a *Q-table* that maps state-action pairs to values indicating the action's "quality", approximating the goal probability or expected reward via the action. The table is updated during simulation runs called *episodes* as

$$Q(s,a) := (1-\alpha) \cdot Q(s,a) + \alpha \cdot (r + \gamma \cdot \max_{\text{action } a' \text{ from } s'} Q(s',a'))$$

for each visited state s, where γ is the discount factor that we set to 1, and α is the *learning rate* hyperparameter that determines the impact of the new information gained during the episode over the previous information in the Q-table. Typically, α is high in the first episodes and then gradually decreases. As the number of episodes goes towards infinity, the Q-table entries approach the optimal values, with the corresponding strategy for state s being to choose $\arg\max_{\text{action } a \text{ from } s} Q(s,a)$. After learning, modes performs an independent SMC analysis under this strategy to estimate the message delivery probability.

Q-learning is popular in machine learning and artificial intelligence applications today, where a neural network stores the Q-table approximately. We work with an explicit Q-table stored in memory. The worst-case memory usage

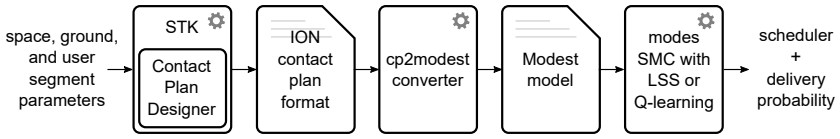


Fig. 8. Satellite routing scheduling toolchain for uncertain DTNs [23]

of Q-learning is thus in $\mathcal{O}(|S| \cdot |A|)$ for state set S and actions A, which is in stark contrast to the otherwise constant memory usage of SMC. The hope is that, in practice, many states either have no choices or there is a small "core" of states [62] reached during learning that suffices to obtain a good strategy.

We adapt Q-learning to distributed strategies by using the *concurrent learning* approach [70]: each agent (in our case: DTN node) learns on its own, keeping and updating its own Q-table and only observing that part of the current state that contains the agent's local information. While straightforward to implement, concurrent learning no longer guarantees convergence and optimality [22]. Its main advantage is that, instead of storing Q-table entries for states from the product state space of all agents, each node's learner only sees the local component state space, so the number of Q-values stored in concurrent Q-learning grows only linearly instead of exponentially with the number of components.

A Modest toolchain for uncertain DTNs. We developed the toolchain outlined in Fig. 8 to convert concrete contact plans (with exact contact timings, obtained from a commercial physics-based modeling environment) into an abstract MODEST MDP model. The generated models follow a simple pattern and are good-for-distribution by construction.

In Listing 4, we show an excerpt from the MODEST model for the example contact plan of Fig. 7. Whenever a node has a contact, the process that models that node contains an `alt` for the three possibilities of sending 1 or 2 copies if available, or listening for incoming data (which may not arrive because either the other node also chose to listen or the data was randomly lost). The model shown is for unreliable communication; we can also generate a reliably communicating variant that uses acknowledgments. Note that the `snd`... and `rcv`... actions do not synchronise: they exist only to make the model more readable, and to identify the choices that the strategy found by LSS or Q-learning makes. Information is exchanged between the `Node`N processes by value passing through transient variables. Listing 4 shows a first model: [24] then implemented many optimisations to that pattern by graph analysis of the contact plan in the cp2modest converter, to make modes execute faster and to remove spurious choices in `alt`s [24, Section 3.2.2]. The latter reduces the space of strategies that LSS samples from, making it more likely to sample good strategies; it also reduces the size of the Q-tables in Q-learning.

```
transient int(0..4) dest1, dest2, dest3, dest4; // destX: target of copies sent by node X
transient int(0..2) data1, data2, data3, data4; // dataX: #copies just sent by node X
action sync;                                    // to synchronise time slots
action snd1to2_1, snd1to2_2, rcv1to2, ...; // only for labelling, no synchronisation
...
process Node2(int(0..2) copies)
{
   alt { // slot 0: contact with 1
   :: when(copies >= 1) snd2to1_1;
      sync palt { :0.9: {= data2 = 1, dest2 = 1, copies -= 1 =} :0.1: {= copies -= 1 =} }
   :: when(copies >= 2) snd2to1_2;
      sync palt { :0.9: {= data2 = 2, dest2 = 1, copies -= 2 =} :0.1: {= copies -= 2 =} }
   :: rcv1to2;
      sync {= 1: copies += dest1 == 2 ? data1 : 0 =}
   };
   alt { // slot 1: contact with 3
   :: when(copies >= 1) snd2to3_1;
      sync palt { :0.9: {= data2 = 1, dest2 = 3, copies -= 1 =} :0.1: {= copies -= 1 =} }
   :: when(copies >= 2) snd2to3_2;
      sync palt { :0.9: {= data2 = 2, dest2 = 3, copies -= 2 =} :0.1: {= copies -= 2 =} }
   :: rcv3to2;
      sync {= 1: copies += dest3 == 2 ? data3 : 0 =}
   };
   sync; // slot 3: no contact
   sync; // slot 4: no contact
   sync  // slot 5: no contact
}
...
par { :: Node1(2) :: Node2(0) :: Node3(0) :: Node4(0) }
```

Listing 4. Excerpt of the MODEST model for the contact plan of Fig. 7

SMC results. We implemented LSS and Q-learning for distributed strategies as described above in modes, and applied this implementation in particular to a realistic LEO Walker constellation of 16 satellites as shown in Fig. 9. In Fig. 10, we show two examples of the results we obtained for this example in [24], comparing LSS and Q-learning (QL) with global and with distributed (local, prefix "L-") information. We show the message delivery probability (SDP) as we vary the message loss probability of all contacts. As a baseline, we used the standard CGR routing algorithm [4,35] that does not take probabilities or multiple message copies into account, and also compared with the model checking-based (L-)RUCoP approach that is implemented in a separate tool using its own input format. In the plots, "Src-Dst" indicates the node numbers of the message source and destination, "Duration"

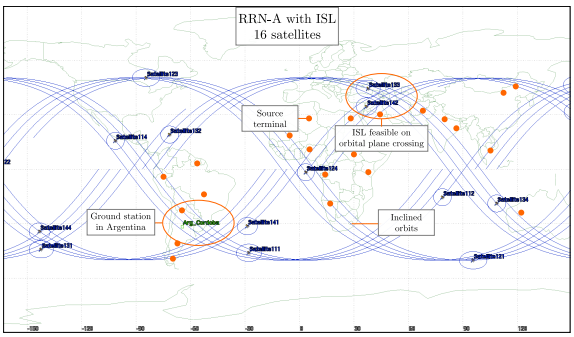


Fig. 9. Walker DTN [24]

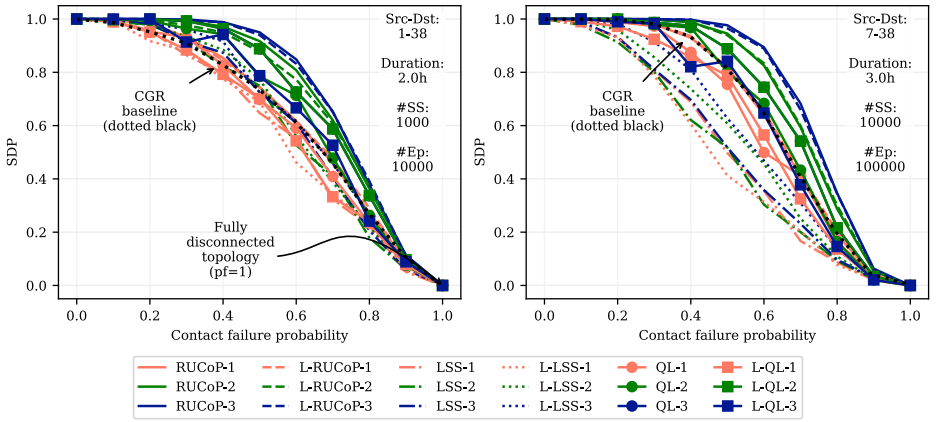


Fig. 10. Message delivery probabilities of best strategies found for the Walker DTN [24]

is the real-time length of the contact plan, "#SS" is the number of strategies sampled in LSS, and "#Ep" is the number of episodes in Q-learning.

We found that L-RUCoP is notably superior to L-LSS for failure probabilities between 0.4 and 0.8. Interestingly, the gap is reduced if we raise the number of schedulers to 10000 in LSS, indicating we are right on the boundary of what can effectively be solved via LSS. Nevertheless, the uninformed sampling strategy of LSS may not be fully adequate for realistic DTN topologies. Q-learning appears to perform better. We also observed that LSS and L-LSS are typically close, but we frequently attain a better probability with distributed strategies. This is likely because of the smaller space of strategies to sample from (see above). We also measured memory usage and runtime, showing that RUCoP's better results come at the expense of significantly higher memory usage and runtime. It needed up to ≈ 20 minutes to terminate, while LSS and Q-learning typically delivered a result in less than a minute. While RUCoP needed as much as 600 MB of memory, LSS consistently used about 100 MB, while Q-learning used slightly more as expected. On some of the other examples we considered in [24], L-RUCoP ran out of memory while L-LSS still delivered results quickly.

Open problems. Both the concrete problem of finding optimal routes for uncertain contact plans as well as the underlying challenge to find practical solutions for distributed-information model checking remain mostly open. LSS and Q-learning are attractive as generic and easy-to-implement approaches that can deliver useful results, but in complex examples, the strategies they find may be far from optimal, and they provide no information about how far these strategies are from the optimum. We would in particular like to investigate modifications to LSS that incorporate more structural information about the current strategy (in contrast to its current opaque integer identifier implementation), and using deep Q-learning methods with neural networks.

Data availability. The models and tools needed to replicate the DTN evaluation described in this section are archived at DOI 10.5281/zenodo.11214677 [25].

5 Optimally Attacking Bitcoin

The Bitcoin [69] cryptocurrency records its transactions in a blockchain to which blocks are added via the proof-of-work principle: participants need to solve a computationally intensive problem to be able to generate or *mine* a valid block. Generally, the first new valid block mined gets appended to the chain, and a certain number of Bitcoins is awarded to the participant that found the block as a reward. However, as a distributed system spanning the globe via the Internet, Bitcoin has to deal with asynchrony: If multiple participants find new blocks at roughly the same time, there are different alternative forks of the Bitcoin blockchain, and a consensus must be reached on which is the valid one. In Bitcoin, the longest available chain is considered the valid one.

Bitcoin as a CTMC. The problem miners have to solve is finding a number that, together with their new block's content, hashes to a value that is smaller than the network's current *difficulty target* value [69, Section 4]. This is done by trying many randomly selected numbers. A try is successful with probability p; then the time until a new block is mined follows a geometric distribution (assuming a constant time t per try, for computing the hash of number and block together). The geometric distribution for ever smaller p and t converges to an exponential distribution, and in practice p and t for Bitcoin are *very* small.

Thus the mining of blocks can abstractly be modelled by a CTMC in which the transition from a chain with n blocks to one with $n+1$ blocks occurs with a certain rate. As the total computational power applied to mining blocks by all miners worldwide (the *hash rate*) changes, the Bitcoin network periodically adjusts the hardness of the problem via the difficulty target value such that the average time to find a new block (the confirmation time) is 10 minutes. In practice the actual confirmation time varies; it was about 12 minutes in 2017 [34]. Thus we use a rate of $\frac{1}{12}$ for the n-to-$n{+}1$-blocks transition in the CTMC.

Attacking trust in Bitcoin. If a large amount of the hash rate (some fraction $M \in [0,1]_{\mathbb{R}}$) is controlled by a malicious party, they could attack the Bitcoin network by secretly working on their own fork until it becomes longer than the "public" one, and then broadcasting the secret fork. These attacks (to Bitcoin but also blockchain in general) are known as *block withholding attacks* and come in several variants. For example, in the double-spending variant a Bitcoin can be spent twice: once on the public fork in block b_i, and once on the secret fork that branches off from publicly known block b_j that is before b_i in the chain. This behaviour can be integrated into an abstract CTMC model of Bitcoin to e.g. compute the expected time until the attack succeeds (i.e. the secret fork becomes longer than the public one by a certain margin) for various values of

M. We built such a model in MODEST and studied similar properties using mcsta and modes in [51].

A more interesting and somewhat easier block withholding attack attempts to undermine the public trust in Bitcoin by just obtaining a longer secret fork from *any* block and then publishing that fork. If done repeatedly, regular users could no longer rely on the persistence of transactions that initially appeared to have become a part of the valid Bitcoin blockchain. In this attack, every time the public fork is extended, the malicious entity may decide between (a) continuing to work on its current secret fork and (b) restarting its secret fork from the new public block. This is because it is no longer necessary to purge a specific block b_i from the public chain as in the double-spending attack.

A case for Markov automata. Due to the presence of the above nondeterministic choice between (a) and (b) to be optimised, this attack can thus no longer be represented in a CTMC model. Our automata formalisms family tree of Fig. 1 contains two direct combinations of CTMCs with the ability to represent nondeterministic choices: CTMDPs and MAs. The latter are a very orthogonal combination of CTMCs and MDPs: They provide two types of transitions—Markovian transitions that execute independently but after a random delay that follows an exponential distribution whose rate is given as part of the transition, and probabilistic transitions that take place immediately, but can synchronise with other probabilistic transitions in parallel composition and lead to a discrete probability distribution over the successor state. This partitioning of the transitions enables parallel composition with action synchronisation without the need to prescribe an ad-hoc operation for combining rates as would be necessary for CTMC or CTMDP. For this reason, MA are preferable for practical modelling, and are supported by MODEST while CTMDPs are not.

Uppaal and Modest models. The attack on trust in Bitcoin was first formally analysed by Fehnker and Chaudhary [34] using SMC with Uppaal SMC. As a consequence of using SMC, they had to run a separate analysis for every possible strategy determining the conditions for when to continue and when to restart, and their results came with a statistical error. Today, Uppaal Stratego [31] may alleviate the former problem.

We instead modelled the same scenario in the MA subset of MODEST, in order to let mcsta find the best strategy directly via PMC and thus without any statistics or the need to run multiple separate analyses. This model is shown in Listing 5. The `HonestPool` process represents the regular Bitcoin miners, who have $100 \cdot (1 - M)\,\%$ of the global hash rate at their disposal and therefore mine a new block every $\frac{12}{1-M}$ minutes. They announce every new block via the `sln` action. The malicious entity's behaviour is defined by the `TrustAttacker` process. They mine new blocks every $\frac{12}{M}$ minutes, and also listen for messages about new blocks from the regular miners. Whenever those find a new block, the `TrustAttacker` is faced with a nondeterministic choice: (a) `cnt`: continue their secret fork, or (b) `rst`: start over from the block just found by the regular

```
const real M;                // fraction of hash rate controlled by malicious pool
const int CD;                // confirmation depth required by victim
const int DB = CD;           // attacker gives up when this far behind
action sln;                  // signal that honest pool mined a new block
action rst;                  // signal that attacker restarts from public fork
action cnt;                  // signal that attacker continues
int(0..CD+1) m_len;          // length of the secret fork
int(-DB..CD+1) m_diff = 0; // length of secret fork minus honest fork
property T_MWinMin = Xmin(T, m_len >= CD && m_diff > 0); // min. exp. time to malicious win
property P_MWinMax = Pmax(<>[T<=2880] (m_len >= CD && m_diff > 0)); // max. prob in 2 days
process HonestPool()
{
   rate(1/12 * (1 - M)) tau; // every 12 / (1 − M) minutes on average:
   sln;                      // honest pool mines a new block
   HonestPool()
}
process TrustAttacker()
{
   do {
   :: rate((1/12) * M)                  // every 12 / M minutes on average:
      {= m_len = min(CD, m_len + 1),    // malicious pool mines a new block
         m_diff++ =}
   :: sln {= m_diff-- =};               // extension of public fork results in
      alt {                             // strategy choice for malicious fork:
      :: rst {= m_len = 0, m_diff = 0 =} // restart (always possible) or
      :: when(m_diff > -DB) cnt         // continue (if not too far behind)
      }
   }
}
par { :: HonestPool() :: TrustAttacker() }
```

Listing 5. MODEST model for optimising the trust attack on Bitcoin [51]

miners. The attacker keeps track of the length of their secret fork since branching off in variable `m_len` and of its length difference compared to the public chain in `m_diff`. The attacker will make its secret fork public once it is (i) longer than the public chain and (ii) at least `CD` blocks long; this latter *confirmation depth* ensures that the new chain is unlikely to be overtaken by other miners.

Model checking results and strategies. We set `M` to 0.2—a hash rate that was at several points in the past achieved by some mining pools—and `CD` to 6, a commonly used value. Our MODEST model specifies two properties of interest: `T_MWinMin` asks for the minimum expected time until the attacker "wins" by publicising their longer chain, while `P_MWinMax` asks for the time-bounded probability that the attacker manages to win within two days (i.e. 2880 minutes). mcsta takes under a second to model-check them with sound algorithms, finding that `T_MWinMin` $= 3736.5920429883377 \pm \epsilon$ and `P_MWinMax` $= 0.5351007861781778 \pm \epsilon$ with the default $\epsilon = 10^{-3}$. That is, the shortest possible expected time for the attack to succeed is 2.6 days, while there is a more than 50/50 chance to succeed within just 2 days. It is thus doubtful whether we should trust Bitcoin if any single actor amasses 20 % or more of the global hashrate.

Now, the above values are *best-case* values for the attacker, if they play the optimal strategy in terms of resetting and continuing their fork. We can let mcsta output this strategy for `T_MWinMin` [51], which results in a file containing 33 state-action decision pairs like

```
+ State: (..., m_len = 5, m_diff = 4)
  Choice: cnt
```

that exhaustively describe the strategy. While 33 textual pairs may still be humanly-interpretable, viz. to understand whether there is some structure or idea to the strategy, this kind of representation is not useful in the general case where models may have millions of states. We thus implemented [14] a connection from mcsta to the dtControl tool [5] that can learn a decision tree from this kind of representation. The resulting tree for this strategy is shown in Fig. 11, and is arguably an explainable and more compact way to present this strategy.

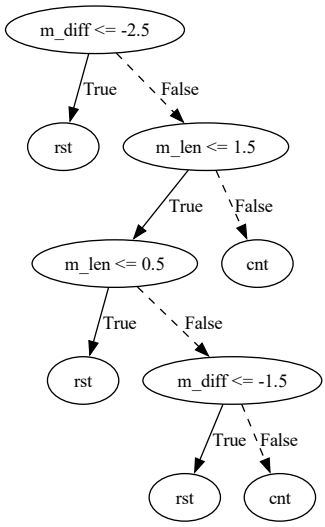


Fig. 11. Decision tree

Open problems. Model checking MA is easy for untimed properties and unbounded expected rewards by running PMC on the embedded MDP. For timed properties, optimal strategies need to know about the remaining time, so dedicated algorithms are needed. For time-bounded reachability, two decent ones with complementary performance exist [19,21]. While these could be improved further—e.g. by developing one algorithm that performs at least as well as either of them on all models—other interesting properties currently lack scalable model checking algorithms: for example, the only available algorithm for time-bounded expected rewards is based on discretisation [56], which drastically exacerbates the state space explosion problem. On the SMC side, LSS does not work for continuous-time strategies out of the box [27], and new clever solutions are needed to make it work beyond simply considering untimed strategies only.

Data availability. The `bitcoin-attack.modest` model presented in this section is part of the QVBS [54], available online at qcomp.org/benchmarks.

6 Conclusion

Different case studies have different needs in terms of conceptual modelling power, modelling language features, and analysis tool capabilities. The Modest language and the Modest Toolset provide the means to easily model and analyse systems containing various quantitative aspects ranging from discrete probabilistic choices to stochastic hybrid behaviour. We highlighted three examples that were modelled in Modest and analysed using different tools from the Modest Toolset: First, in the case of **power supply noise in a NoC**,

the simple formalism of DTMC was sufficient. For the detailed concrete model, however, the MODEST language feature of declaring and using one's own complex data types was very helpful. PMC via mcsta was the analysis method of choice, however significant effort was needed to abstract the model until it became tractable for PMC due to the state space explosion problem. Second, for **routing in satellite constellations**, nondeterministic choices needed to be modelled, and optimised over by the analysis tool. Here, MDP fit the problem very well with their ability to model decision-making under uncertainty. We auto-generated MODEST models from contact plans computed by domain-specific software. Due to the need to find implementable routing strategies in the distributed-information setting of satellite constellations, we could not use PMC; instead, we adapted the LSS and Q-learning approaches to allow SMC to handle both distributed information and nondeterminism. Finally, to **optimally attack Bitcoin**, we showed that MA fit the problem well due to the combination of the stochastic time-to-next-block with the nondeterministic choices between continuing and restarting the secret fork. Using PMC with mcsta again, we were able to compute an optimal strategy with little computational effort and present it in a compact and explainable manner as a decision tree generated via a new connection to dtControl.

References

1. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Trans. Model. Comput. Simul. **28**(1), 6:1–6:39 (2018). https://doi.org/10.1145/3158668
2. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) Hybrid Systems. Lecture Notes in Computer Science, vol. 736, pp. 209–229. Springer (1992). https://doi.org/10.1007/3-540-57318-6_30
3. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
4. Araniti, G., Bezirgiannidis, N., Birrane, E., Bisio, I., Burleigh, S., Caini, C., Feldmann, M., Marchese, M., Segui, J., Suzuki, K.: Contact graph routing in DTN space networks: overview, enhancements and performance. IEEE Comms. Magazine **53**(3), 38–46 (2015). https://doi.org/10.1109/MCOM.2015.7060480
5. Ashok, P., Jackermeier, M., Kretínský, J., Weinhuber, C., Weininger, M., Yadav, M.: dtControl 2.0: Explainable strategy representation via decision tree learning steered by experts. In: Groote, J.F., Larsen, K.G. (eds.) 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 12652, pp. 326–345. Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_17
6. Baier, C.: Probabilistic model checking. In: Esparza, J., Grumberg, O., Sickert, S. (eds.) Dependable Software Systems Engineering, NATO Science for Peace and Security Series – D: Information and Communication Security, vol. 45, pp. 1–23. IOS Press (2016). https://doi.org/10.3233/978-1-61499-627-9-1
7. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook

of Model Checking, pp. 963–999. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_28
8. Baier, C., Katoen, J.P.: Principles of model checking. MIT Press (2008)
9. Bellman, R.: A Markovian decision process. Journal of Mathematics and Mechanics **6**(5), 679–684 (1957)
10. Bohnenkamp, H.C., D'Argenio, P.R., Hermanns, H., Katoen, J.P.: MoDeST: A compositional modeling formalism for hard and softly timed systems. IEEE Trans. Software Eng. **32**(10), 812–830 (2006). https://doi.org/10.1109/TSE.2006.104
11. Bohnenkamp, H.C., Hermanns, H., Katoen, J.P., Klaren, R.: The Modest modeling tool and its implementation. In: Kemper, P., Sanders, W.H. (eds.) 13th International Conference on Computer Performance Evaluations, Modelling Techniques and Tools (TOOLS). Lecture Notes in Computer Science, vol. 2794, pp. 116–133. Springer (2003). https://doi.org/10.1007/978-3-540-45232-4_8
12. Bonet, B., Geffner, H.: Labeled RTDP: Improving the convergence of real-time dynamic programming. In: Giunchiglia, E., Muscettola, N., Nau, D.S. (eds.) 13th International Conference on Automated Planning and Scheduling (ICAPS). pp. 12–21. AAAI (2003)
13. Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Automated compositional importance splitting. Sci. Comput. Program. **174**, 90–108 (2019). https://doi.org/10.1016/j.scico.2019.01.006
14. Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Digging for decision trees: A case study in strategy sampling and learning. In: International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA) (2024), submitted, under review.
15. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: An efficient statistical model checker for nondeterminism and rare events. Int. J. Softw. Tools Technol. Transf. **22**(6), 759–780 (2020). https://doi.org/10.1007/s10009-020-00563-2
16. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: Quantitative model and tool interaction. In: Legay, A., Margaria, T. (eds.) 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 10206, pp. 151–168 (2017). https://doi.org/10.1007/978-3-662-54580-5_9
17. Budde, C.E., Hartmanns, A., Klauck, M., Kretínský, J., Parker, D., Quatmann, T., Turrini, A., Zhang, Z.: On correctness, precision, and performance in quantitative verification (QComp 2020 competition report). In: Margaria, T., Steffen, B. (eds.) 9th International Symposium on Leveraging Applications of Formal Methods (ISoLA). Lecture Notes in Computer Science, vol. 12479, pp. 216–241. Springer (2020). https://doi.org/10.1007/978-3-030-83723-5_15
18. Bulychev, P.E., David, A., Larsen, K.G., Mikucionis, M., Poulsen, D.B., Legay, A., Wang, Z.: Uppaal-SMC: Statistical model checking for priced timed automata. In: Wiklicky, H., Massink, M. (eds.) 10th Workshop on Quantitative Aspects of Programming Languages and Systems (QAPL). EPTCS, vol. 85, pp. 1–16 (2012). https://doi.org/10.4204/EPTCS.85.1
19. Butkova, Y., Fox, G.: Optimal time-bounded reachability analysis for concurrent systems. In: Vojnar, T., Zhang, L. (eds.) 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 11428, pp. 191–208. Springer (2019). https://doi.org/10.1007/978-3-030-17465-1_11
20. Butkova, Y., Hartmanns, A., Hermanns, H.: A Modest approach to Markov automata. ACM Trans. Model. Comput. Simul. **31**(3), 14:1–14:34 (2021). https://doi.org/10.1145/3449355

21. Butkova, Y., Hatefi, H., Hermanns, H., Krcál, J.: Optimal continuous time Markov decisions. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) 13th International Symposium on Automated Technology for Verification and Analysis (ATVA). Lecture Notes in Computer Science, vol. 9364, pp. 166–182. Springer (2015). https://doi.org/10.1007/978-3-319-24953-7_12
22. Claus, C., Boutilier, C.: The dynamics of reinforcement learning in cooperative multiagent systems. In: Mostow, J., Rich, C. (eds.) 15th National Conference on Artificial Intelligence and 10th Innovative Applications of Artificial Intelligence Conference (AAAI, IAAI). pp. 746–752. AAAI Press / The MIT Press (1998)
23. D'Argenio, P.R., Fraire, J.A., Hartmanns, A.: Sampling distributed schedulers for resilient space communication. In: Lee, R., Jha, S., Mavridou, A. (eds.) 12th International NASA Formal Methods Symposium (NFM). Lecture Notes in Computer Science, vol. 12229, pp. 291–310. Springer (2020). https://doi.org/10.1007/978-3-030-55754-6_17
24. D'Argenio, P.R., Fraire, J.A., Hartmanns, A., Raverta, F.: Comparing statistical, analytical, and learning-based routing approaches for delay-tolerant networks. ACM Trans. Model. Comput. Simul. (2024), to appear.
25. D'Argenio, P.R., Fraire, J.A., Hartmanns, A., Raverta, F.: Comparing statistical, analytical, and learning-based routing approaches for delay-tolerant networks (artifact). Zenodo (2024). https://doi.org/10.5281/zenodo.11214677, to appear.
26. D'Argenio, P.R., Hartmanns, A., Legay, A., Sedwards, S.: Statistical approximation of optimal schedulers for probabilistic timed automata. In: Ábrahám, E., Huisman, M. (eds.) 12th International Conference on Integrated Formal Methods (iFM). Lecture Notes in Computer Science, vol. 9681, pp. 99–114. Springer (2016). https://doi.org/10.1007/978-3-319-33693-0_7
27. D'Argenio, P.R., Hartmanns, A., Sedwards, S.: Lightweight statistical model checking in nondeterministic continuous time. In: Margaria, T., Steffen, B. (eds.) 8th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA). Lecture Notes in Computer Science, vol. 11245, pp. 336–353. Springer (2018). https://doi.org/10.1007/978-3-030-03421-4_22
28. D'Argenio, P.R., Jeannet, B., Jensen, H.E., Larsen, K.G.: Reduction and refinement strategies for probabilistic analysis. In: Hermanns, H., Segala, R. (eds.) Second Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM-PROBMIV). Lecture Notes in Computer Science, vol. 2399, pp. 57–76. Springer (2002). https://doi.org/10.1007/3-540-45605-8_5
29. D'Argenio, P.R., Legay, A., Sedwards, S., Traonouez, L.M.: Smart sampling for lightweight verification of Markov decision processes. Int. J. Softw. Tools Technol. Transf. **17**(4), 469–484 (2015). https://doi.org/10.1007/S10009-015-0383-0
30. D'Argenio, P.R., Wolovick, N., Terraf, P.S., Celayes, P.: Nondeterministic labeled Markov processes: Bisimulations and logical characterization. In: 6th International Conference on Quantitative Evaluation of Systems (QEST). pp. 11–20. IEEE Computer Society (2009). https://doi.org/10.1109/QEST.2009.17
31. David, A., Jensen, P.G., Larsen, K.G., Mikucionis, M., Taankvist, J.H.: Uppaal Stratego. In: Baier, C., Tinelli, C. (eds.) 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 9035, pp. 206–211. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_16
32. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer,

S. (eds.) 23rd International Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 6806, pp. 349–355. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_27
33. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: 25th Annual IEEE Symposium on Logic in Computer Science (LICS). pp. 342–351. IEEE Computer Society (2010). https://doi.org/10.1109/LICS.2010.41
34. Fehnker, A., Chaudhary, K.: Twenty percent and a few days – optimising a Bitcoin majority attack. In: Dutle, A., Muñoz, C.A., Narkawicz, A. (eds.) 10th International NASA Formal Methods Symposium (NFM). Lecture Notes in Computer Science, vol. 10811, pp. 157–163. Springer (2018). https://doi.org/10.1007/978-3-319-77935-5_11
35. Fraire, J.A., De Jonckère, O., Burleigh, S.C.: Routing in the space Internet: A contact graph routing tutorial. Journal of Network and Computer Applications **174**, 102884 (2021). https://doi.org/10.1016/j.jnca.2020.102884
36. Fränzle, M., Hahn, E.M., Hermanns, H., Wolovick, N., Zhang, L.: Measurability and safety verification for stochastic hybrid systems. In: Caccamo, M., Frazzoli, E., Grosu, R. (eds.) 14th ACM International Conference on Hybrid Systems: Computation and Control (HSCC). pp. 43–52. ACM (2011). https://doi.org/10.1145/1967701.1967710
37. Frehse, G.: PHAVer: algorithmic verification of hybrid systems past HyTech. Int. J. Softw. Tools Technol. Transf. **10**(3), 263–279 (2008). https://doi.org/10.1007/S10009-007-0062-X
38. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. Int. J. Softw. Tools Technol. Transf. **15**(2), 89–107 (2013). https://doi.org/10.1007/S10009-012-0244-Z
39. Giro, S., D'Argenio, P.R.: Quantitative model checking revisited: Neither decidable nor approximable. In: Raskin, J.F., Thiagarajan, P.S. (eds.) 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS). Lecture Notes in Computer Science, vol. 4763, pp. 179–194. Springer (2007). https://doi.org/10.1007/978-3-540-75454-1_14
40. Giro, S., D'Argenio, P.R.: On the expressive power of schedulers in distributed probabilistic systems. Electron. Notes Theor. Comput. Sci. **253**(3), 45–71 (2009). https://doi.org/10.1016/j.entcs.2009.10.005
41. Haddad, S., Monmege, B.: Reachability in MDPs: Refining convergence of value iteration. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) 8th International Workshop on Reachability Problems (RP). Lecture Notes in Computer Science, vol. 8762, pp. 125–137. Springer (2014). https://doi.org/10.1007/978-3-319-11439-2_10
42. Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. Theor. Comput. Sci. **735**, 111–131 (2018). https://doi.org/10.1016/J.TCS.2016.12.003
43. Hahn, E.M., Hartmanns, A.: A comparison of time- and reward-bounded probabilistic model checking techniques. In: Fränzle, M., Kapur, D., Zhan, N. (eds.) Second International Symposium on Dependable Software Engineering: Theories, Tools, and Applications (SETTA). Lecture Notes in Computer Science, vol. 9984, pp. 85–100 (2016). https://doi.org/10.1007/978-3-319-47677-3_6
44. Hahn, E.M., Hartmanns, A.: Symblicit exploration and elimination for probabilistic model checking. In: Hung, C.C., Hong, J., Bechini, A., Song, E. (eds.) 36th ACM/SIGAPP Symposium on Applied Computing (SAC). pp. 1798–1806. ACM (2021). https://doi.org/10.1145/3412841.3442052

45. Hahn, E.M., Hartmanns, A., Hensel, C., Klauck, M., Klein, J., Kretínský, J., Parker, D., Quatmann, T., Ruijters, E., Steinmetz, M.: The 2019 comparison of tools for the analysis of quantitative formal models (QComp 2019 competition report). In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) 25 Years of TACAS: TOOLympics. Lecture Notes in Computer Science, vol. 11429, pp. 69–92. Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_5
46. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.P.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods Syst. Des. **43**(2), 191–232 (2013). https://doi.org/10.1007/s10703-012-0167-z
47. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: iscasMc: A web-based probabilistic model checker. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) 19th International Symposium on Formal Methods (FM). Lecture Notes in Computer Science, vol. 8442, pp. 312–317. Springer (2014). https://doi.org/10.1007/978-3-319-06410-9_22
48. Hartmanns, A.: An overview of Modest models and tools for real stochastic timed systems. In: Dubslaff, C., Luttik, B. (eds.) 5th Workshop on Models for Formal Analysis of Real Systems (MARS@ETAPS). EPTCS, vol. 355, pp. 1–12 (2022). https://doi.org/10.4204/EPTCS.355.1
49. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In: Ábrahám, E., Havelund, K. (eds.) 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 8413, pp. 593–598. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_51
50. Hartmanns, A., Hermanns, H.: Explicit model checking of very large MDP using partitioning and secondary storage. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) 13th International Symposium on Automated Technology for Verification and Analysis (ATVA). Lecture Notes in Computer Science, vol. 9364, pp. 131–147. Springer (2015). https://doi.org/10.1007/978-3-319-24953-7_10
51. Hartmanns, A., Hermanns, H.: A Modest Markov automata tutorial. In: Krötzsch, M., Stepanova, D. (eds.) 15th International Reasoning Web Summer School. Lecture Notes in Computer Science, vol. 11810, pp. 250–276. Springer (2019). https://doi.org/10.1007/978-3-030-31423-1_8
52. Hartmanns, A., Hermanns, H., Berrang, P.: A comparative analysis of decentralized power grid stabilization strategies. In: Rose, O., Uhrmacher, A.M. (eds.) Winter Simulation Conference (WSC). pp. 158:1–158:13. WSC (2012). https://doi.org/10.1109/WSC.2012.6465083
53. Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. In: Lahiri, S.K., Wang, C. (eds.) 32nd International Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 12225, pp. 488–511. Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_26
54. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: Vojnar, T., Zhang, L. (eds.) 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 11427, pp. 344–350. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_20
55. Hartmanns, A., Sedwards, S., D'Argenio, P.R.: Efficient simulation-based verification of probabilistic timed automata. In: 2017 Winter Simulation Conference (WSC). pp. 1419–1430. IEEE (2017). https://doi.org/10.1109/WSC.2017.8247885
56. Hatefi-Ardakani, H.: Finite horizon analysis of Markov automata. Ph.D. thesis, Saarland University, Germany (2017), http://scidok.sulb.uni-saarland.de/volltexte/2017/6743/

57. Hensel, C., Junges, S., Katoen, J.P., Quatmann, T., Volk, M.: The probabilistic model checker Storm. Int. J. Softw. Tools Technol. Transf. **24**(4), 589–610 (2022). https://doi.org/10.1007/S10009-021-00633-Z
58. Howard, R.A.: Dynamic Programming and Markov Processes. MIT Press (1960)
59. Kamali, M., Katoen, J.P.: Probabilistic model checking of AODV. In: Gribaudo, M., Jansen, D.N., Remke, A. (eds.) 17th International Conference on Quantitative Evaluation of Systems (QEST). Lecture Notes in Computer Science, vol. 12289, pp. 54–73. Springer (2020). https://doi.org/10.1007/978-3-030-59854-9_6
60. Klauck, M., Hermanns, H.: A Modest approach to dynamic heuristic search in probabilistic model checking. In: Abate, A., Marin, A. (eds.) 18th International Conference on Quantitative Evaluation of Systems (QEST). Lecture Notes in Computer Science, vol. 12846, pp. 15–38. Springer (2021). https://doi.org/10.1007/978-3-030-85172-9_2
61. Köhl, M.A., Klauck, M., Hermanns, H.: Momba: JANI meets Python. In: Groote, J.F., Larsen, K.G. (eds.) 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 12652, pp. 389–398. Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_23
62. Kretínský, J., Meggendorfer, T.: Of cores: A partial-exploration framework for Markov decision processes. Log. Methods Comput. Sci. **16**(4) (2020), https://lmcs.episciences.org/6833
63. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) 23rd International Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 6806, pp. 585–591. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_47
64. Kwiatkowska, M.Z., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. Formal Methods Syst. Des. **29**(1), 33–78 (2006). https://doi.org/10.1007/s10703-006-0005-2
65. Kwiatkowska, M.Z., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theor. Comput. Sci. **282**(1), 101–150 (2002). https://doi.org/10.1016/S0304-3975(01)00046-9
66. Lanotte, R., Merro, M., Munteanu, A.: A Modest security analysis of cyber-physical systems: A case study. In: Baier, C., Caires, L. (eds.) 38th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE). Lecture Notes in Computer Science, vol. 10854, pp. 58–78. Springer (2018). https://doi.org/10.1007/978-3-319-92612-4_4
67. Legay, A., Sedwards, S., Traonouez, L.M.: Scalable verification of Markov decision processes. In: Canal, C., Idani, A. (eds.) 4th Workshop on Formal Methods in the Development of Software (WS-FMDS). Lecture Notes in Computer Science, vol. 8938, pp. 350–362. Springer (2014). https://doi.org/10.1007/978-3-319-15201-1_23
68. Lewis, B., Hartmanns, A., Basu, P., Shridevi, R.J., Chakraborty, K., Roy, S., Zhang, Z.: Probabilistic verification for reliable network-on-chip system design. In: Larsen, K.G., Willemse, T.A.C. (eds.) 24th International Conference on Formal Methods for Industrial Critical Systems (FMICS). Lecture Notes in Computer Science, vol. 11687, pp. 110–126. Springer (2019). https://doi.org/10.1007/978-3-030-27008-7_7
69. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008), https://bitcoin.org/bitcoin.pdf

70. Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. Auton. Agents Multi Agent Syst. **11**(3), 387–434 (2005). https://doi.org/10.1007/s10458-005-2631-2
71. Quatmann, T., Katoen, J.: Sound value iteration. In: Chockler, H., Weissenbacher, G. (eds.) 30th International Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 10981, pp. 643–661. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_37
72. Raverta, F.D., Fraire, J.A., Madoery, P.G., Demasi, R.A., Finochietto, J.M., D'Argenio, P.R.: Routing in delay-tolerant networks under uncertain contact plans. Ad Hoc Networks **123**, 102663 (2021). https://doi.org/10.1016/j.adhoc.2021.102663
73. Roberts, R., Lewis, B., Hartmanns, A., Basu, P., Roy, S., Chakraborty, K., Zhang, Z.: Probabilistic verification for reliability of a two-by-two network-on-chip system. In: Lluch-Lafuente, A., Mavridou, A. (eds.) 26th International Conference on Formal Methods for Industrial Critical Systems (FMICS). Lecture Notes in Computer Science, vol. 12863, pp. 232–248. Springer (2021). https://doi.org/10.1007/978-3-030-85248-1_16
74. Rubino, G., Tuffin, B. (eds.): Rare Event Simulation using Monte Carlo Methods. Wiley (2009). https://doi.org/10.1002/9780470745403
75. Sproston, J.: Decidable model checking of probabilistic hybrid automata. In: Joseph, M. (ed.) 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT). Lecture Notes in Computer Science, vol. 1926, pp. 31–45. Springer (2000). https://doi.org/10.1007/3-540-45352-0_5
76. Sutton, R.S., Barto, A.G.: Reinforcement learning – An introduction. Adaptive computation and machine learning, MIT Press (1998)
77. Watkins, C.J.C.H., Dayan, P.: Q-learning. Mach. Learn. **8**, 279–292 (1992). https://doi.org/10.1007/BF00992698