# Statistical Model Checking for Distributed Probabilistic-Control Hybrid Automata with Smart Grid Applications

João Martins        André Platzer        João Leite

2011

CMU-CS-11-119

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

## Abstract

This technical report is a more detailed version of a published paper [12].

The power industry is currently moving towards a more dynamical, intelligent power grid. This *Smart Grid* is still in its infancy and a formal evaluation of the expensive technologies and ideas on the table is necessary before committing to a full investment. In this paper, we argue that a good model for the Smart Grid must match its basic properties: it must be hybrid (both evolve over time, and perform control/computation), distributed (multiple concurrently executing entities), and allow for asynchronous communication and stochastic behaviour (to accurately model real-world power consumption). We propose Distributed Probabilistic-Control Hybrid Automata (DPCHA) as a model for this purpose, and extend Bounded LTL to Quantified Bounded LTL in order to adapt and apply existing statistical model-checking techniques. We provide an implementation of a framework for developing and verifying DPCHAs. Finally, we conduct a case study for Smart Grid communications analysis.

# 1 Introduction

The ultimate promise of the Smart Grid is that of a more stable, energy-efficient, adaptable, secure, resilient power grid, while delivering cheaper electricity. Currently, energy consumption follows fairly predictable patterns that need to be very closely matched by power generation (otherwise blackouts or damage to the infrastructure may occur). There are peak hours (e.g., people arrive home on a hot summer day and turn on the AC), and low hours (e.g., during the night). Certain power generators run permanently at 100% capacity, providing support to what is known as the base load. More adaptable but more expensive generators change their output to match demand, varying the price of energy throughout the day. During peak hours, it might be necessary to turn on highly adaptable and expensive peak load generators, making energy extremely expensive for those few hours.

One of the core ideas of the Smart Grid is that generators will no longer passively adapt to consumption. Instead, power consumers both at the lower level (e.g., appliances such as washing machines) and higher level (utilities serving some geographical area) will feed their desired consumption back into the Grid. Indeed, utilities and home-owners have already begun deploying smart meters and appliances that make available more detailed, up-to-date energy consumption information. This gives the smarter Grid better foresight, increasing its robustness and its ability to reschedule non-critical appliances (e.g., the dishwasher) to off-peak hours, reducing energy costs.

Given the size and criticality of the power infrastructure, it is clear that Smart Grid technologies have to be analysed very carefully. Furthermore, the cost of providing real test-beds for all technologies is prohibitive, especially if the infrastructure can sustain damage when things go wrong. Formal verification, on the other hand, allows us to study a *model* of the system in question, sidestepping the above issues. Given an appropriate model, we may then check the system for desirable properties: how much can be saved with appliance rescheduling? Do Smart Meters help in predicting and optimizing load? Does that prediction help balance load across generators? Not only are the answers to these questions useful in furthering our understanding of the technologies, they also give us hints about how they may be improved upon before real-world deployment.

For the above reasons, we believe that formal verification of new technologies is fundamental for the Smart Grid. The first step in this endeavour is to find adequate models for the Grid. The models need to be flexible and generic so they can be reused for multiple projects and ideas, yet match the nature and properties of the Grid. Trying to force the use of models that do not fit the properties of the Grid leads to modelling idiosyncrasies, effectively making modelling and verification much harder than they need to or should be (humanly and computationally).

What, then, are the properties of the Smart Grid? Most importantly, it is a *cyber-physical* system. Its infrastructure exists in the real-world and follows the laws of physics (e.g., a generator increasing its output over time), but it also contains control components that make decisions and change the state instantaneously. Thus, it is a hybrid system, i.e., has both continuous and discrete dynamics. Some mathematical models and verification techniques for hybrid systems extend automata theory by allowing continuous evolution to occur in each state (e.g., [10, 6, 1]), but verification is known to be undecidable for most cases [6]. Differential Dynamic Logic ($d\mathcal{L}$) allows the specification of both the properties and the behaviour of a hybrid system [14] and provides a proof calculus for verification.

Another property of the Smart Grid is that it is *distributed*. The Grid is not one monolithic system, but composed of a large number of distributed and *communicating* entities, from the power generators down to electrical substations to the utilities, households, appliances and meters. All of these elements coexist, communicate and cooperate with one another in real-time. Most automata models support the notion of composition, allowing a fixed number of automata to execute concurrently. The Grid, however, is *dynamically* distributed: appliances are turned on and off, power lines can be cut, and meters may fail. The model must allow entities/elements to enter, leave and communicate as part of the system dynamics. Dynamic I/O Automata [9] allow a dynamic number of elements, but are not hybrid. Quantified Differential Dynamic Logic ($Qd\mathcal{L}$) [15] also allows a dynamic number of elements and is hybrid, but like Dynamic I/O automata, has a shared-memory communication model. Proposals for Smart Grid communication are currently based on IP and message-passing protocols, so that forcing a fundamentally different shared-memory paradigm causes an oversimplification of a critical component of the Grid.

Finally, the Grid exhibits *stochastic behaviour*. As we have seen, power consumption follows dynamic but well-understood patterns. Using non-determinism to model these patterns fails to encode the information that we have regarding "normal" Grid behaviour. Hybrid system models are generally non-deterministic, attempting to verify safety properties that hold even in the worst-case. In the Grid, worst-case scenarios (e.g., all lines cut at the same time, all appliances always on) are sure to bring about a complete collapse, and most safety properties *will not* hold! Alternatively, a probabilistic model enables 1) a more detailed and accurate representation of the Grid's consumption patterns and 2) a more comprehensive quantitative study. Since we know most interesting properties are not always true, we may *estimate the probability* that they hold. This precludes $Qd\mathcal{L}$ [15], which is not stochastic, whereas Stochastic $d\mathcal{L}$ [16] is not distributed. I/O Automata have a stochastic extension, but it is not distributed [9].

Petri Nets are inherently dynamically distributed and there have been stochas-

tic and differential extensions [17, 5]. However, the notion of markings flowing place to place is not one that we find when designing the participants of the Smart Grid. These participants would be composed of multiple markings scattered over different places of the Petri graph. Several entities of the same class (e.g., microwaves), sharing the same "control" graph, would create a multitude of markings superimposed in the same places. Markings would have to be associated with one another to keep track of the entities as a whole, instead of considering an entity as an indivisible structure. There is also no immediately available communication mechanism for transmitting messages (with a payload). In conclusion, while many Petri Net variants feature mechanisms very similar to those of the Smart Grid, it is our belief their actual implementation is generally differs enough to warrant the Grid a model of its own.

In [13] the state of the system is given by a composition of objects and messages. All objects evolve continuously as long as no invariant is violated, and fire probabilistic discrete transitions when they are. Asynchronous communication is achieved by assigning a delivery time to all messages upon creation. The decision to do a discrete or continuous transition depends exclusively on whether an invariant is violated, making the dynamics of this model very restricted.

In summation, we need a model that is composed of many different entities. These entities should be able to enter and leave the system at will, representing failures and appliances being turned on or off. Furthermore, the entities must be able to communicate asynchronously: given the scale of the Grid and the impact of message delivery delays, it is unrealistic to assume synchronous (instant) communication. Finally, the system must be able to behave probabilistically, in order to encode uncertain environments, e.g., power consumption. To the best of our knowledge, no existing model naturally incorporates all of these properties.

In this paper, we propose Distributed Probabilistic-Control Hybrid Automata (DPCHA) as such a model. We take care that the system can be easily sampled from (to obtain execution traces), with the objective of applying existing efficient statistical verification techniques.

Previous work has shown that statistical model checking (SMC) is a promising approach for the verification of probabilistic systems [4, 2, 3, 20, 7, 18, 8, 22, 21]. Given a property and a model, SMC techniques will repeatedly sample traces from the model and check if they satisfy the property. Every new result provides more information on whether the property holds for arbitrary traces. While known to be unsound, SMC can arbitrarily approximate the *probability* that the property holds very efficiently, making many otherwise intractable problems accessible.

Logics traditionally used in the specification of properties for these hybrid systems generally consider a fixed state-space. This makes them insufficient for the representation of properties of distributed systems. We propose Quantified

3

Bounded Linear Temporal Logic, an extension to Bounded Linear Temporal Logic [22, 7, 4, 18] that handles the dynamic state space of DPCHA using quantification over the elements of the system. A similar phenomenon has been studied in the context of Java threads [19], for example, but not for cyber-physical systems.

The main contribution of this paper is the proposal of a model that naturally adapts to Smart Grid scenarios and for which these techniques are applicable, enabling meaningful studies of the system.

We present some technical background in Section 2, and our DPCHA model in Section 3. To specify properties we define QBLTL in Section 4. We briefly explain Bayesian statistical model checking in Section 5, and develop an initial case study in the Smart Grid domain in Section 6. We conclude in Section 7.

## 2 Preliminaries

Before developing the distributed model, we will begin by introducing how a single entity behaves (e.g., microwave, generator). Thus, we briefly recall discrete-time hybrid automata (DTHA) as described in [22]. Each entity must have a *state*, e.g., current and desired power output of a generator. The entity is in a *location* that specifies how the state should *flow* as time passes, e.g., spooling up generator to match desired output. Finally, the entity may decide to *jump* from one location to another, e.g., the microwave switches to "defrost". We refer to an entity's *situation* as the pair of its location and state. Thus, DTHA are hybrid because they allow continuous evolution (time passing) and discrete transitions between locations.

**Definition 1** (DTHA). *A discrete-time hybrid automaton consists of*

- $\langle Q, E \rangle$, *a "control graph" with $Q$ as* locations *and $E \subseteq Q \times Q$ as the* edges

- $\mathbb{R}^n$ *is the state space of the automaton's state*

- $jump_e : \mathbb{R}^n \rightharpoonup \mathbb{R}^n$, *a partial function defining how the state changes when jumping along edge $e$*

- $\varphi_q : \mathbb{R}_{\geq 0} \times \mathbb{R}^n \to \mathbb{R}^n$, *flows. $\varphi_q(t; x)$ is the result of a continuous evolution at location $q \in Q$ after time $t$ when starting in state $x \in \mathbb{R}^n$*

- $(q_0; x_0) \in Q \times \mathbb{R}^n$, *an initial situation*

Suppose an entity is in a situation $(q; x)$. It may jump along an edge $e$ originating from $q$, updating its state according to $jump_e$. Or it may remain in $q$ for some time $t$, updating its state according to the flow $\varphi_q$ (which can be, for instance, the solution of a differential equation system). Since there might be multiple options for the next step, the automaton is non-deterministic.

4

**Definition 2** (Transition relation for DTHA). *The transition relation for a DTHA is defined as:*

$$(q; x) \xrightarrow{\alpha} (\overline{q}; \overline{x}), \text{ where}$$

- *When $\alpha = t \in \mathbb{R}_{\geq 0}$ is a time, then $(q; x) \xrightarrow{t} (q; \overline{x})$ iff $\overline{x} = \varphi_q(t; x)$*

- *When $\alpha = e \in E$ is an edge from $q$ to $\overline{q}$, then $(q; x) \xrightarrow{e} (\overline{q}; \overline{x})$ iff $\overline{x} = jump_e(x)$*

It is only possible to jump along an edge $e$ if the entity's state is in the domain of $jump_e$. In this case, we say that $e$ is *enabled* and that $jump_e$ works as a *guard* for $e$. A *scheduler* $\delta : Q \times \mathbb{R}^n \to \mathbb{R}_{\geq 0} \cup E$ that, given a situation $(q; x)$, decides the next action $\alpha$ (a flow or an edge to jump), can be applied repeatedly to an entity's situation to obtain a *trace* for that entity.

We now have defined the dynamics of a single entity. To make it behave probabilistically, all we need to do is to make $\delta : Q \times \mathbb{R}^n \to \mathcal{D}(E \cup \mathbb{R}_{\geq 0})$ return a *probability distribution* over all possible actions instead of a single action $\alpha$. Sampling from this distribution gives the entity its next step.

## 3 Distributed Probabilistic Control Hybrid Automata

A single entity's behaviour is given by its control graph, flows and jumps. We have mentioned that traditional notions of automata composition are not dynamic enough for the Grid. *Configuration automata* [9] keep track of multiple executing entities (also automata) that can enter and leave the system, resulting in two layers of automata that have no particular intuition in the Grid. Furthermore, the automata force communication to be immediate and synchronous.

Instead, like in Petri Nets, we assume the control graphs for all types of entities are given, so we understand how microwaves (for example) behave, but are not required to know how many are present in the system. With the control graphs, jumps and flows for each type of entity (e.g., microwave, meter), it is trivial to encode the behaviour of all types of entities in one global control graph.

The global control graph accommodates several entities of each type, not unlike Petri Net markings. These entities are characterised by their situation and execute like a single entity from the previous section. This is the basic intuition for the notions of Distributed Discrete-Time Hybrid Automata (DDTHA) and Distributed Probabilistic Control Hybrid Automata (DPCHA) that we define below.

To maintain a sensible global notion of time, the DDTHA will do a continuous transition only if all executing entities decide to do so. If any entity decides to do a jump, then the other entities must either jump as well or *flounder* (i.e. doing a

discrete transition with no effect), keeping time unchanged. In this sense, discrete transitions take precedence over continuous transitions, but consume no time.

We must still address the ability to communicate and to allow entities to enter or exit the system. We reduce these two concerns (communication and dynamic number of entities) to five elementary *actions*: $\texttt{new}[N]$, $\texttt{die}$, $\texttt{snd}[l][T]$, $\texttt{recv}[l][R]$, $\texttt{jmp}$. Each edge in the control graph features an action. When an entity jumps along that edge, its action is executed. $\texttt{jmp}$ is a null action so that the entity simply follows $\texttt{jump}_e$. $\texttt{new}[N]$ additionally creates a new entity with a situation specified by a function $N$, and $\texttt{die}$ makes the jumping entity exit the system. $\texttt{snd}[l][T]$ and $\texttt{recv}[l][R]$ send and receive messages through a *channel* determined by function $l$. The content of the sent message is given by function $T$, whereas the receiving entity's state is updated according to $R$, taking into account both its current state and message content. To achieve asynchronous communication, sent messages are stored at the global automaton level in a "buffer", and are removed later when received. Thus, each action is characterised by functions determining exactly how it is executed. We let $\mathcal{A}_e$ denote the action of edge $e$, which happens in addition to the effects of $\texttt{jump}_e$.

In summary, the control graph retains its general structure, but annotates each edge with actions for communication and dynamism. Instead of a single initial situation, we have an initial set of active entities and an initial situation for each entity. Active entities all evolve time-synchronously, each following the rules of DTHA. As entities jump along edges, they execute the associated actions, enabling communication and complex interactions.

**Definition 3** (DDTHA). *A Distributed Discrete-Time Hybrid Automaton is composed of*

- $\mathbb{R}^n$, *the state space for each entity, with $n \in \mathbb{N}$*

- $\mathbb{R}^{n_m}$, *the state space of each message's content, with $n_m \in \mathbb{N}$*

- $\mathcal{A} = \{\texttt{new}[N], \texttt{die}, \texttt{snd}[l][T], \texttt{recv}[l][R], \texttt{jmp}\}$, *the set of all actions, with channel specification functions $l : \mathbb{R}^n \to C$, new entity creation functions $N : \mathbb{R}^n \to Q \times \mathbb{R}^n$, message transmission functions $T : \mathbb{R}^n \to \mathbb{R}^{n_m}$ and message reception functions $R : \mathbb{R}^{n_m} \times \mathbb{R}^n \to \mathbb{R}^n$*

- $\langle Q, E \rangle$, *control graph with locations $Q$ and edges $E \subseteq Q \times \mathcal{A} \times Q$*

- $\texttt{jump}_e : \mathbb{R}^n \times \mathbb{R}^n$, *a relation when $\mathcal{A}_e = \texttt{recv}$ or function $\texttt{jump}_e : \mathbb{R}^n \to \mathbb{R}^n$ otherwise, defining acceptable state updates when jumping along edge $e$*

- $\varphi_q$, *as in Def. 1*

- *L: a (countable) set of entity identifiers*

- $A_0 \subset L$: *a finite set of initial* active entities

- $S_0 : A_0 \to Q \times \mathbb{R}^n$, *a function with a* situation *for each initial active entity*

- *C, a (countable) set of communication channels*

The state of a DDTHA consists of the situations of all its entities, active and past. Information about past entities is maintained, which is necessary for the well-definedness property checking. The state also maintains a set of "in transit" messages (sent but not received), enabling asynchrony of communication.

**Definition 4** (State of a DDTHA). *The state of a DDTHA is given by* $AS = (A, S, M)$ *with*

- $A \subset L$ *a finite set of the labels of active entities*

- $S : L \rightharpoonup Q \times \mathbb{R}^n$, *a partial function with the situation of active/past entities*

- $M \subseteq C \times \mathbb{R}^{nm}$, *a set of unreceived messages and respective channels*

One interesting issue arises when an entity $a$ decides (through a scheduler) to flow for $t$ time units but sometime at $t' < t$ some other entity $b$ finishes its own flow and schedules a jump. In this situation, the DDTHA also schedules a discrete transition, but $a$ cannot be allowed to reevaluate its previous decision to flow for $t$ time (e.g., the washing machine should not stop because someone turned on the TV across the country). Therefore, we assume without loss of generality that each entity stores in its state (e.g., in its first coordinate) how long it must still flow, denoted by $\delta$-time. In state $(A, S, M)$, an entity $a$'s $\delta$-time is denoted $\delta$-time$^S(a)$. When the DDTHA schedules a discrete transition, any entity with non-zero $\delta$-time will flounder, thus only truly rescheduling once its flow decision finishes executing.

Another important element of discrete transitions is message reception. There must be an injective mapping from "in transit" messages to receiving entities so that they get exactly one message. Injection ensures each receiving entity gets at most one message. Of course, the entities must react accordingly, and received/sent messages are removed/added to $M$.

The following example justifies our choice of probabilities and asynchronous communication and illustrates a simplified modelling of the Smart Grid.

**Example 1.** *Newest generation smart meters feed up-to-date information into the Grid, including power consumption from the appliance level up to substation and utility levels and so on; the Grid also needs to match generator output with power consumption. There is a Grid control infrastructure that maintains this fragile balance.*



**Figure 1:** Simplified Smart Grid

*An ideal model for this scenario would have entities representing appliances, consuming energy, shutting off and powering on, and sending messages into the Grid through channel $l_c$. Another (unique) entity, called the Power Controller (PC), would react to messages from $l_c$ and control generator output by sending it messages through $l_t$. Unfortunately, it is computationally infeasible to model every appliance in a country-wide Grid (except maybe Monaco or Nauru!). A sensible simplification instead represents* classes *of appliances that get turned on at around the same time for a very similar duration, like ACs/computers in offices, the TVs at home, etc. Ideally, the exact times and durations are given by probability distributions, simulating real-world behaviour. We obtain a much more manageable number of entities by using* classes *of appliances instead of* individual *appliances.*

*Real networks become congested so that messages are not delivered instantly. Probabilities can be used to simulate this delay: the PC's choice for flow time, for instance, could be given as a normal random variable Normal(5, $\epsilon_{pc}$). The PC then waits around 5s before getting the message. We may assign its individual scheduler a 0.8 probability of jumping along a* recv *edge, so that there is a 0.2 chance it will be further delayed (simulating message loss and retransmission or congestion). These remaining* 0.2 *can be split between sending control messages to generators or deciding to do another flow.*

*To justify our choice of asynchronous communication, suppose that whenever an appliance class entity sends a message, the PC is forced to ignore its $\delta$-time and synchronise to receive that message. The PC is deviating from its original specification of continuous evolution according to a Normal RV, which would make the semantics, meaning and usefulness of the model unclear.*

The transition relation for DDTHA relies heavily on that for DTHA, in that single entities will behave according to Definition 2. All this information is encoded in the following definition.

**Definition 5** (Transition for DDTHA). *The transition relation of a DDTHA is defined inductively as*

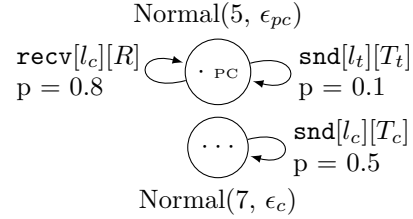$$(A, S, M) \xrightarrow{\alpha} (\overline{A}, \overline{S}, \overline{M})$$

*where A is non-empty, $\alpha \in \mathbb{R}_{\geq 0} \cup (A \to (E \cup \mathbb{R}_{\geq 0} \cup \{F\}))$, iff*

- *If $\alpha = t \in \mathbb{R}_{\geq 0}$, then $\forall_{a \in A} \, a \in \overline{A}$, $S(a) \xrightarrow{t} \overline{S}(a)$, $\delta\text{-time}^{\overline{S}}(a) = \delta\text{-time}^{S}(a) - t$ and $\delta\text{-time}^{\overline{S}}(a) \geq 0$*

- *If $\alpha = \tau : A \to E \cup \mathbb{R}_{\geq 0} \cup \{F\}$, then there are partial injective mappings $\mu_c : \{(c, \mathbb{R}^{nm}) \in M\} \to \{a \in A : \tau(a) = (q, \mathtt{recv}[l][R], \overline{q})\}$ from messages of each channel to entities scheduled to receive on that channel, $\overline{M} = (M \setminus \bigcup \{range(\mu_c) : c \in C\}) \cup \{(l(S(a)), T(S(a))) : a \in A, \tau(a) = (q, \mathtt{snd}[l][T], \overline{q})\}$ and $\forall_{a \in A}$ if $\delta\text{-time}^{S}(a) > 0$, then $\tau(a) = F$, $a \in \overline{A}$ and $S(a) = \overline{S}(a)$; otherwise if $\delta\text{-time}^{S}(a) = 0$*

    - *If $\tau(a) = t \in \mathbb{R}_{\geq 0}$, then $S(a) = \overline{S}(a)$ except $\delta\text{-time}^{\overline{S}}(a) = t$*

    - *If $\tau(a) = (q, \mathtt{jmp}, \overline{q}) \in E$, then $a \in \overline{A}$ and $S(a) \xrightarrow{(q,\overline{q})} \overline{S}(a)$*

    - *If $\tau(a) = (q, \mathtt{new}[N], \overline{q}) \in E$, then $a \in \overline{A}$, $S(a) \xrightarrow{(q,\overline{q})} \overline{S}(a)$, and there exists a completely new $\overline{a} \notin A$, $\overline{a} \in \overline{A}$ such that $\overline{S}(\overline{a}) = N(S(a))$*

    - *If $\tau(a) = (q, \mathtt{die}, \overline{q}) \in E$, then $a \notin \overline{A}$ and $S(a) \xrightarrow{(q,\overline{q})} \overline{S}(a)$*

    - *If $\tau(a) = (q, \mathtt{snd}[l][T], \overline{q}) \in E$, then both $a \in \overline{A}$, $S(a) \xrightarrow{(q,\overline{q})} \overline{S}(a)$ and $(l(S(a)), T(S(a))) \in \overline{M}$*

    - *If $\tau(a) = (q, \mathtt{recv}[l][R], \overline{q}) \in E$, then $\mu_c(a) = (c, y) \notin \overline{M}$ with $c = l(S(a))$, $a \in \overline{A}$ and $(S(a), R(y, S(a)) \in \mathtt{jump}_{\tau(a)}$ and $\overline{S}(a) = R(y, S(a))$*

There may be multiple messages to deliver to an entity, and vice-versa. To remove this source of non-determinism, we simply use a combination of lexicographical and temporal ordering to choose a single assignment.

Given a single-entity scheduler $\delta : Q \times \mathbb{R}^n \to \mathbb{R}_{\geq 0} \cup E$ like those of DTHA, we define a DDTHA scheduler $\Delta$ for an automaton state $(A, S, M)$ as follows

1. If $\forall_{a \in A} \, \delta(S(a)) \in \mathbb{R}$, then $\Delta(AS) = \min\{\delta(S(a)) : a \in A\}$

2. If $\exists_{a \in A} \, \delta(S(a)) \in E$, then $\Delta(AS) = \tau$, where for each $a \in A$:
$$\tau(a) = \begin{cases} \delta(S(a)) & , \delta\text{-time}^{S}(a) = 0 \\ F & , \delta\text{-time}^{S}(a) > 0 \end{cases}$$

This definition specifies global state behaviour from each entity's individual behaviour. It encodes the previously explained idea that the system will only evolve over time as long as all entities wish to do so. As soon as one wishes to jump, all

entities that are not commited to continuous evolution will be allowed to jump. The others will flounder, specified by action $F$.

Each scheduler $\Delta$ yields a single execution of the system. These valid executions are called traces, and are formalised as follows.

**Definition 6** (Trace of a DPCHA). *A* trace *of a DPCHA is a sequence* $\sigma = (AS_0, t_0)$, $(AS_1, t_1)$, *..., with $AS_i$ as in Def. 4, $t_i \in \mathbb{R}_{\geq 0}$ such that 1) $AS_0 = (A_0, S_0, \emptyset)$ and 2) for each $i \in \mathbb{N}_{>0}$ (up to the size of the trace if it is finite):*

1. $AS_{i-1} \xrightarrow{\Delta(AS_{i-1})} AS_i$

2. $t_{i-1} = \begin{cases} \Delta(AS_{i-1}) & \text{, if } \Delta(AS_{i-1}) \in \mathbb{R}_{\geq 0} \\ 0 & \text{, if } \Delta(AS_{i-1}) \in (L \to E \cup \{F\}) \end{cases}$

Given the priority of discrete transitions, we make the assumption of *divergence of time*, i.e., we do not consider schedulers whose traces have infinitely many transitions in finite time. This ensures there is no infinite sequence of jumps, i.e. that time actually passes and the system evolves.

To obtain DPCHA, we probabilise the single entity scheduler $\delta$, from which the global scheduler $\Delta$ is obtained. In effect, we sample from each entity's distribution sequentially until all entities have decided on their course of action. From this set of actions we construct the global action, and the distribution of the global DPCHA action is derived from the distribution of the entities' actions. This results in *Distributed Probabilistic-Control Hybrid Automata*, allowing us to formally specify the model in Example 1.

## 4 Quantified Bounded Linear Temporal Logic

The next step towards applying (Bayesian) SMC techniques [22, 7] is to define a way to specify properties and to check whether they are satisfied by the execution traces of the system. These properties must deal with the distributed nature of the Grid. For example, we want to be able to aggregate power demand, or how much power is being generated in total.

We start from Bounded Linear Temporal Logic (BLTL) [22, 7, 4, 18], featuring a strong *bounded until* $\mathbf{U}^t$ operator to deal with time. $\phi_1 \mathbf{U}^t \phi_2$ states that $\phi_1$ must hold until $\phi_2$ holds and $\phi_2$ holds before the time bound $t$. It does not require $\phi_1$ to hold when $\phi_2$ first holds, but it does require $\phi_2$ to hold at some point before $t$.

It has been proven that BLTL formulae can be checked with only finite traces as long as the system guarantees divergence of time [22]. Unfortunately, BLTL lacks the capability to express properties about a system with a dynamic number of

entities, and existing alternatives are domain-specific or bounded [19]. Each entity contains its variables (e.g., refrigerator temperature), but to refer to those variables we must first get a handle on the entity itself. We do this by allowing for quantification over active entities in the system (i.e. actualist quantification). Similarly, we allow any computable aggregation function to range over the entities and return some aggregate value (e.g., $\max$, $\sum$). This results in Quantified Bounded linear Temporal Logic, whose syntax is defined as follows:

**Definition 7** (Syntax of QBLTL). *Formulae of QBLTL are given by the following grammar, with $* \in \{+, -, \div, \times, \hat{\ }\}$ and $\sim \in \{\leq, \geq, =\}$:*

$$\theta ::= c \mid \theta_1 * \theta_2 \mid \pi_i(e) \mid \mathsf{E}(e) \mid \boldsymbol{ag}[e](\theta), \text{ with } i \in \mathbb{N}, c \in \mathbb{Q}$$
$$\phi ::= \mathsf{E}(e) \mid \theta_1 \sim \theta_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi_1 \mid \phi_1 \, \boldsymbol{U}^t \phi_2 \mid \exists e.\phi_1$$

In the above, $e$ is a variable denoting an entity. $\pi_i(e)$ is the $i$th variable of entity $e$. $\boldsymbol{ag}[e](\theta)$ stands as a template for *any* computable, associative and commutative aggregation function (e.g., $\sum[e](\pi_{\text{temp}}(e))$). We abuse notation to define $\mathsf{E}(e)$ as 1) an indicator function for whether $e$ is active, 2) formula evaluating to true iff $e$ is active. This is useful for filtering out entities in aggregations and specifying properties quantifying over entities that exit the system. For example, $\sum[e](\mathsf{E}(e))$ evaluates to the number of active entities in the automaton.

As usual, we define the other logical operators from Def. 7, e.g., $\phi_1 \wedge \phi_2 \equiv \neg(\neg\phi_1 \vee \neg\phi_2)$, and temporal operators such as $\mathbf{F}^t\phi \equiv \text{true } \mathbf{U}^t\phi$ (eventually $\phi$ holds before $t$) and $\mathbf{G}^t\phi \equiv \neg\mathbf{F}^t\neg\phi$ ($\phi$ always holds until $t$).

The semantics of QBLTL are given with respect to traces and a variable assignment $\eta : Vars(\phi) \rightarrow L$ to entity labels (cf. Def. 3), where $Vars(\phi)$ is the set of variables occurring in $\phi$. $\eta$ is used to keep track of which entity variables refer to, as in first order logic.

Let $\sigma = (AS_0, t_0), (AS_1, t_1), ...$ be a trace of a DPCHA. We define that trace $\sigma$ and assignment $\eta$ satisfy a formula $\phi$ by a relation $\sigma, \eta \models \phi$. Let $\sigma^i$ be the trace suffix of $\sigma$ starting at position $i$, e.g., $\sigma^0 = \sigma$ and $\sigma^k = (AS_k, t_k), (AS_{k+1}, t_{k+1}), ...$ Let $\llbracket \theta \rrbracket^\eta_{\sigma^k}$ represent the value of interpreting $\theta$ under $AS_k$ and assignment $\eta$, and $AS_i = (A_i, S_i, M_i)$ for all $i \geq 0$.

**Definition 8** (Semantics of QBLTL). *The semantics of QBLTL for a trace $\sigma^k = (AS_k, t_k), (AS_{k+1}, t_{k+1}), ...$ are defined by the interpretation of terms:*

- $\llbracket c \rrbracket^\eta_{\sigma^k} = c$,

- $\llbracket \theta_1 * \theta_2 \rrbracket^\eta_{\sigma^k} = \llbracket \theta_1 \rrbracket^\eta_{\sigma^k} * \llbracket \theta_2 \rrbracket^\eta_{\sigma^k}$, *interpreting the syntactic operator \* by the corresponding semantic operator \*,*

- $\llbracket \pi_i(e) \rrbracket^\eta_{\sigma^k} = x_i$, *where $S_k(\eta(e)) = (q; x) \in Q \times \mathbb{R}^n$, and $x_i$ is the projection to the $i$th coordinate of $x$,*

11

- $[\![\mathsf{E}(e)]\!]^{\eta}_{\sigma^k} = 1$ *if $\eta(e) \in A_k$ and $0$ otherwise.*

- $[\]\!]^{\eta}_{\sigma^k} = \boldsymbol{ag}\left([\![\theta]\!]^{\eta\{e \mapsto l_1\}}_{\sigma^k}, \boldsymbol{ag}\left(..., [\![\theta]\!]^{\eta\{e \mapsto l_n\}}_{\sigma^k}\right)\right)$*, where $(l_1, l_2, ..., l_n)$ is some ordering of $A_k$ (the result is independent of the ordering since $\boldsymbol{ag}$ is associative and commutative),*

*and the following relation:*

- $\sigma^k, \eta \models \mathsf{E}(e)$ *iff $\eta(e) \in A_k$*

- $\sigma^k, \eta \models \theta_1 \sim \theta_2$ *iff $[\![\theta_1]\!]^{\eta}_{\sigma^k} \sim [\![\theta_2]\!]^{\eta}_{\sigma^k}$, extending the syntactic comparison operator $\sim$ to the corresponding semantic $\sim$,*

- $\sigma^k, \eta \models \phi_1 \vee \phi_2$ *iff $\sigma^k, \eta \models \phi_1$ or $\sigma^k, \eta \models \phi_2$,*

- $\sigma^k, \eta \models \neg\phi_1$ *iff $\sigma^k, \eta \not\models \phi_1$ or it is false that $\sigma^k, \eta \models \phi_1$,*

- $\sigma^k, \eta \models \phi_1 \boldsymbol{U}^t \phi_2$ *iff there exists $i \in \mathbb{N}$ such that 1) $\sum_{l=0}^{i} t_{k+l} \leq t$, 2) for all $j$ such that $0 \leq j < i$, $\sigma^{k+j}, \eta \models \phi_1$ and 3) $\sigma^{k+i}, \eta \models \phi_2$,*

- $\sigma^k, \eta \models \exists e.\phi_1$ *iff there exists $l \in A_k$ such that $\sigma^k, \eta\{e \mapsto l\} \models \phi_1$*

As usual in logic, $\sigma^k, \eta \models \phi$ is only well-defined if $\eta$ contains an assignment for every free variable of $\phi$. In $\exists e$, $e$ is a variable ranging over *currently existing* entities. However, these entities may leave the system in the future, leaving us with a "dangling" variable. We illustrate this next.

**Example 2.** *Consider a model where a consumer entity is created whenever an appliance is turned on, and that disappears when it is turned off. While verifying this model we may want to check that some appliances are always running at high power, e.g., a refrigerator with a consumption minimum of 300 watts. This property can be expressed in the following QBLTL formula $\exists e.\boldsymbol{G}^{24h}\pi_{consumption}(e) \geq 300$.*

*Given a trace for a sample day, we attempt to evaluate the formula. For instance, suppose $e$ represents a washing machine that is running at first, but finishes its program and leaves the active Grid sometime later. What is the meaning of $\pi_{consumption}(e) \geq 300$ after the washing machine leaves the system?*

*The actualist semantics that we chose achieve what we believe is a good compromise that avoids semantic pitfalls, in the same vein as [15]. The key is to keep track of past entities' state in $S$ so that the semantics are well-defined even with exiting entities. The main point, however, is that the special predicate/term $\mathsf{E}(\cdot)$ can be used to handle entities that have left the system. The property above should have been $\exists e.\boldsymbol{G}^{24h}\mathsf{E}(e) \wedge \pi_{consumption}(e) \geq 300$, i.e. is there an entity that is permanent and that is always consuming above 300.*

We have made sure that our extensions are compatible with earlier SMC approaches [22, 7] so that we can lift the theory of SMC directly to our scenario. First, we guarantee that finite simulations are sufficient for checking whether a QBLTL formula is satisfied, because we cannot run infinite simulations. Due to our setting, this is a straightforward extension of results from [22]. We define a bound $\#(\phi)$ of a QBLTL formula by having $\#(\theta) = 0$ for any term $\theta$. For any other logical connective excluding the until operator (e.g., $\neg\phi_1, \phi_1 \vee \phi_2$), we define the bound as the maximum of the bound of its direct subformulae, e.g., $\#(\phi_1 \vee \phi_2) = \max(\#(\phi_1), \#(\phi_2))$, and $\#(\exists e.\phi) = \#(\phi)$. Finally, $\#(\phi_1 \, \mathbf{U}^t \phi_2) = t + \max(\#(\phi_1), \#(\phi_2))$. It can now be shown that $\phi$ is satisfied by two infinite traces as long as the prefixes bounded by $\#(\phi)$ of those traces are the same.

**Lemma 1** (QBLTL has bounded simulation traces)**.** *Let $\phi$ be a QBLTL formula and $k \in \mathbb{N}$. Then for any two infinite traces $\sigma = (AS_0, t_0), (AS_1, t_1), ...$ and $\overline{\sigma} = (\overline{AS}_0, \overline{t}_0), (\overline{AS}_1, \overline{t}_1), ...$ with $AS_{k+I} = \overline{AS}_{k+I}$ and $t_{k+I} = \overline{t}_{k+I}$, for all $I \in \mathbb{N}$ with $\sum_{0 \leq l < I} t_{k+l} \leq \#(\phi)$ we have that $\sigma^k \models \phi$ iff $\overline{\sigma}^k \models \phi$.*

The proof is done by induction on QBLTL formulae. The original proof for BLTL [22] extends directly to our additions. It then follows that sampling can be bounded with $\#(\phi)$.

**Lemma 2** (Bounded sampling)**.** *The problem $\sigma \models \phi$ is well defined and can be checked for QBLTL formulae $\phi$ and traces $\sigma$ based only on a finite prefix of $\sigma$ of bounded duration.*

Again, thanks to our compatible setting, the proof for this lemma lifts directly from [22]. Without this result, SMC would not be applicable in our scenario.

## 5 Bayesian Statistical Model Checking

Statistical Model Checking [4, 2, 3, 20, 7, 18, 22, 21] is a conceptually simple technique that has received attention due to its application to many practical situations. We follow the presentation of a Bayesian approach to the method closely, as presented in [22].

SMC tries to determine the probability $p$ that an arbitrary trace of an automaton satisfies a QBLTL formula $\phi$. Two core Bayesian approaches have been proposed: interval estimation and hypothesis testing. These methods diverge from the traditional model checking problem in that a trace that does not satisfy a formula $\phi$ is not a counter-example, but instead *evidence* that $p < 1$. For simplicity, we describe the hypothesis testing algorithm in [22] and refer there for an interval estimation algorithm.

The hypothesis testing algorithm attempts to solve the problem "is the probability that property $\phi$ holds greater or equal to $\theta$", also represented as $P_{\geq\theta}\phi$. That is, we compare the null hypothesis $H_0 : p \geq \theta$ with the alternate hypothesis $H_1 : p < \theta$. We can represent the result of each sampled trace satisfying $\phi$ by Bernoulli random variables with the real probability $p$. After $n$ samples, we have $d = \{x_1, ..., x_n\}$ draws from those Bernoulli RV's, and each result gives us further evidence either for $H_0$ or for $H_1$. Since these hypothesis are mutually exclusive, we can assume that the prior probabilities add to 1, $P(H_0) + P(H_1) = 1$. Bayes' theorem gives us the posterior probabilities as $P(H_i|d) = \frac{P(d|H_i)P(H_i)}{P(d)}$ with $i \in \{0, 1\}$, for every $d$ with $P(d) = P(d|H_0)P(H_0) + P(d|H_1)P(H_1) > 0$, which is always the case in this instance.

**Definition 9** (Bayes factor). *The Bayes factor $\mathcal{B}$ of sample $d$ and hypotheses $H_0$, $H_1$ is $\frac{P(d|H_0)}{P(d|H_1)}$.*

The value of the Bayes factor as defined above, obtained from data $d$ by sampling and testing the property, can be seen as evidence in favour of the acceptance of hypothesis $H_0$. The inverse $\frac{1}{\mathcal{B}}$, on the other hand, is evidence in favour of $H_1$. We can then choose a threshold $T$ for how much evidence is required before we accept one of the hypotheses.

From [22], we know an efficient way to calculate the Bayes factor for $H_0$, $H_1$:

$$\mathcal{B}_n = \frac{1 - \int_\theta^1 g(u)du}{\int_\theta^1 g(u)du} \left( \frac{1}{F_{(x+\alpha, n-x+\beta)}(\theta)} - 1 \right),$$

in the case of beta priors, where $x$ is the number of successes in the draws $d = (x_1, ..., x_n)$ and $F_{(s,t)}(\cdot)$ is the Beta distribution function with parameters $s, t$. The actual algorithm can be found in Figure 2.

The algorithm samples traces from the DPCHA, then checks them against the given formula $\phi$. Since the result of these checks can be seen as drawing from a Bernoulli RV with the desired probability, the algorithm then uses the Bayes factor to calculate how much evidence is in favour of either $H_0$ or $H_1$. The amount of evidence changes with each new draw, resulting in an algorithm that adapts termination to the amount of information it can extract at each iteration. Eventually, enough evidence is amassed for one of the hypotheses, and it is accepted. More details about this and a more sophisticated *estimation* algorithm (that we use in the following) can be found in [22].

**Input:** DPCHA automaton $A$, QBLTL property $\phi$, probability $\theta$, threshold $T \geq 1$ and Beta prior density $g$ for unknown parameter $p$

$n := 0$                                       {// Total number of traces drawn}

$x := 0$                                    {// Total number of traces satisfying $\phi$}

**loop**

    $\sigma :=$ sample trace from DPCHA $A$     {// using probabilistic $\Delta$, see Sect. 3}

    $n := n + 1$

    **if** $\sigma \models \phi$ (see Def. 8) **then**

        $x := x + 1$

    **end if**

    $\mathcal{B} :=$ BayesFactor$(n, x)$

    **if** $\mathcal{B} > T$ **then**

        **return** $H_0$ accepted

    **else if** $\mathcal{B} < \frac{1}{T}$ **then**

        **return** $H_1$ accepted

    **end if**

**end loop**

**Figure 2:** Bayesian statistical model checking for hypothesis testing

# 6 Case Study: Smart Grid

We now develop a case study using a simplified model of Smart Grid every day operations. We show the versatility of our model, how smoothly it fits to the verification methods defined previously, and how easily SMC can be used to check important properties. Recall that the Smart Grid is a fusion of the Power Grid and the Cyber Grid. The hope is that communication capabilities and direct feedback from the consumer level will allow the Smart Grid to provide energy more efficiently and cost-effectively. We use the techniques implemented in our framework to study what properties of the communications layer of the Grid are important for achieving this goal. We focus on the trade-offs between cost-relevant parameters of the network and overall system performance and safety.

## 6.1 Model Description

Our model features several key components of the Smart Grid. It considers consumers (refrigerators, computers, etc), which are responsible for the power demand. On the other side, generators will produce the necessary power. The Smart Grid control infrastructure is also represented as a single entity. It is responsible for aggregating power output and consumption, and making the control decisions that

keep them close together to avoid blackouts and power spikes. Finally, the model uses a consumer controller. While such an entity does not exist in the real world, it is necessary here to organise the appearance of consumers.

### 6.1.1 Consumers

We will represent consumers by classes of appliances as in Example 1. This simplification is what makes our model computationally accessible. It is, however, innocuous: it simply does away with overly complex modelling details of no interest to the high-level functioning of the Grid. We do not care about the exact number of computers in use at a given point; we do care about the overall power consumption of office, or home, appliances since they get turned on and then off at roughly the same times. This type of behaviour, which explains the predictable patterns of modern energy consumption, is typically bell-like. This represents all these appliances being gradually turned on, reaching a peak energy demand, and then their gradually shutting off and demanding less and less power. We also impose optional limits to maximum consumption and maximum consumption increase (i.e. its derivative). All the parameters that define this behaviour can easily be changed to tweak the model for specific behaviours.

To simulate the feedback that Smart Meters send up the Grid, each consumer will decide whether or not to send a message with its current power demand. This decision is made at intervals given by a random variable $\delta \sim \text{Normal}(5, 3)$. The probability of sending a message increases with the time since the last sent message. Messages sent by consumers, or classes of appliances, can be interpreted as messages by local substations sending aggregated power consumption data to the Grid control infrastructure.

Finally, if power consumption for a given entity is becoming very close to zero (i.e. most appliances in this class have already shut down), then the consumer announces it will leave the system and removes itself entirely. Consumers converge to zero only asymptotically in order to keep smooth, continuous behaviour. We only remove them when consumption is close enough to zero not to make a difference in global consumption values.

### 6.1.2 Generators

The model contains one generator that produces the Grid's power output. The generator has one goal power output, set by the power infrastructure to be slightly above current power consumption. It also has detailed capabilities, such as how fast it can decrease or increase power output, as well as its maximum acceleration (and negative acceleration). These parameters can be used to encode the varying

degrees of flexibility (and cost!) of different types of generators.

The generator will always use maximum and minimum acceleration to reach its desired output. Similarly to the consumer entities, generators periodically send messages containing their current output to the Grid control infrastructure. If the generator decides not to send a message, it receives control messages with 0.98 probability. This simulates a 0.02 probability of message loss. These control messages contain the desired power output, as decided by the Grid, and to which the generator must immediately adapt.

The implementation of the generators and consumers actually makes certain control decisions (such as when to stop accelerating power output) while doing continuous evolution. Since these decisions are purely local and do not affect other components of the system, the simplification has no impact on the model's behaviour.

### 6.1.3 Power Controller

The most important component of the model is the Power Controller (PC). The PC does control loops every $\delta \sim \text{Normal}(5, 1)$ and makes decisions that keep the Grid within satisfactory operational bounds (by keeping the difference between power consumption and production below a given threshold). During these loops, the PC receives messages both from the consumers and from the generators with 0.98 probability. The content of these messages is used to compute consumption and total production values. Comparison between these two makes the PC decide whether or not to send the generator a control message asking for it to spool up or down. Notice that the PC only receives messages during the control loop, which cannot interrupt continuous evolution. Therefore, these messages are not received at the same time they are sent, and are actually in transit for some time.

In more complex scenarios with multiple generators, the PC will have to implement a policy to decide which generator(s) will be asked to spool up or down in response to change in power demand. It will have to weigh adaptability and cost with the risk of not adapting fast enough to cope with the Grid's fickle demands during peak-time. Thus, the PC's importance cannot be underestimated. It is, in fact, the brain of the Smart Grid.

### 6.1.4 Consumer Controller

The Grid's consumption patterns are emulated by the Consumer Controller (CC). The CC responsible for spawning consumer entities with different parameters such that global consumption behaviour mimics that of the real Grid. To do this, it

executes a control loop with the "usual interval". During this loop, the CC's role is to decide whether to create a new consumer or not, up to a maximum of 20 consumers. This decision is made by flipping a biased coin whose probability changes with the time of day. Of course, there is a higher probability of appliances being turned on in the morning than, say, in the middle of the night. Table 1 shows these probabilities.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|-----|------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.2 | 0.05 | 0.07 | 0.1 | 0.1 | 0.2 | 0.1 | 0.2 | 0.1 | 0.2 | 0.1 | 0.2 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 0.1 | 0.2 | 0.2 | 0.1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 1:** Probability of creating a new consumer entity as a function of the time of day

Not only that, but consumers will behave differently depending on when they are created. Consumers appearing during the night or late evening require less energy but last longer (from seven to eight hours), and could represent street-lights for example. Consumers appearing during the day usually need more energy and last for only two to three hours, but have a higher probability of appearing.

### 6.1.5 Communication Overview

In this model, there are three main communication flows:

1. Consumers periodically send their consumption to the PC. This is meant to represent the feedback loop of the Smart Grid. At the household level, Smart Meters tally and send consumption information to the substations, which aggregate all these demands, and in turn send them to the control infrastructure. Consumer messages in our model represent this aggregated data.

2. Generators do much the same as consumers: they report their output so that it can be considered by the PC.

3. Finally, the PC sends control messages to the generators in order to control Grid power output.

While flow 1. abstracts away the communication between Smart Meters and the substation (recall that this is what makes the model computationally accessible), 2. and 3. are a fairly accurate representation of actual communication flow in the Grid. This is actually the way the Grid is controlled, although there are additional geographic considerations in the real-world. As has been mentioned, current

18

Smart Grid proposals use IP networks - a kind of parallel Internet - for the Grid. This matches our model's message-sending abstraction, and the delays in receiving messages are also accounted for by message reception not interrupting continuous flow.

The global system, which can clearly be identified as the composition of different "types" of entities, can be seen in Figure 3.
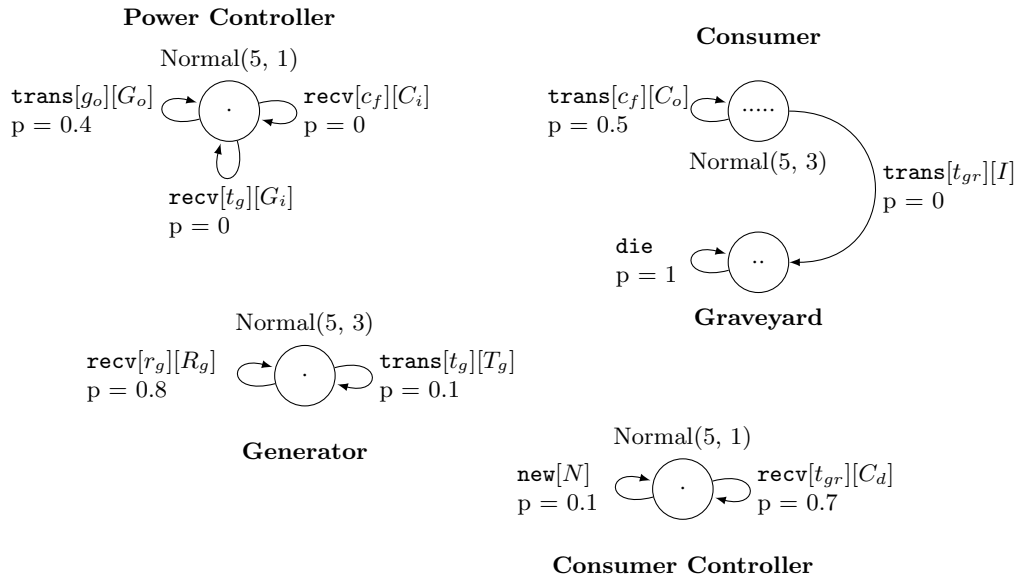


**Figure 3:** Smart Grid model

Figure 4 shows aggregate power consumption, generator output, the PC's estimated consumption and the number of active consumer entities during one day. The shape of sample curves matches the patterns observed in reality, with peak times and a break for lunch. This indicates that our model, even simplified, simulates reasonable Grid behavioural patterns. While certain traces do not result in curves of this particular shape, they still exhibit features found in the Smart Grid such as high peaks, a base load, and both gradual and sudden ascents and descents.

## 6.2    Network Performance Evaluation

In this section we wish to investigate what the impact of network reliability on the system level properties is by checking how resilient the Grid is to message loss.
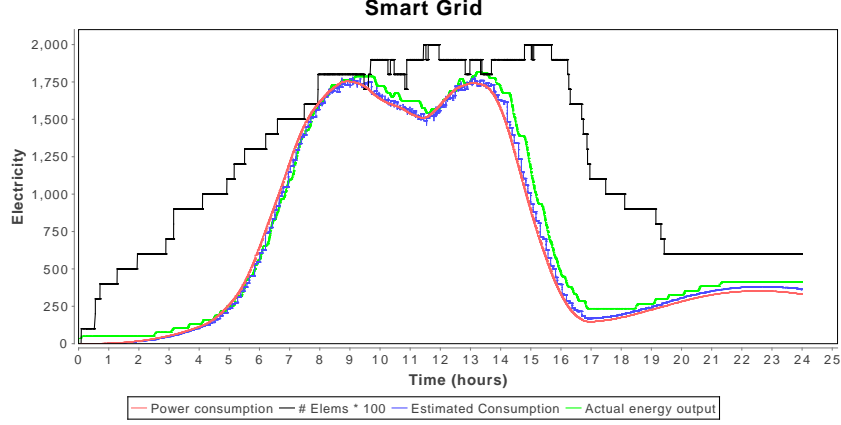
**Figure 4:** Smart Grid scenario with one generator

We use a benchmark of two core properties. Property (1) states that the output of the generator is always within 400 units of the actual demand within a 24h period. This can be encoded in QBLTL as:

$$\mathbf{G}^{24h}\left|\sum[e]\left(Gen(e)\pi_{\text{output}}(e)\right) - \sum[e]\left(Cons(e)\pi_{\text{consumption}}(e)\right)\right| < 400$$

Property (2) states that the PC's estimate of power consumption is not too far from the actual value. The PC's variables 0 through 19 store the consumer's last consumption values (as given by their last messages). In QBLTL,

$$\mathbf{G}^{24h}\left|\sum[e]\left(Gen(e)\pi_{\text{output}}(e)\right) - \sum[e]\left(PC(e)\left(\pi_0(e) + ... + \pi_{19}(e)\right)\right)\right| < 250$$

Here, *PC*, *Gen* and *Cons* are simply indicator functions for whether the element is the power controller, a generator or a consumer.

Property (2) is a pre-requisite for property (1). Without a good estimation of power output, the PC can never request the correct changes to the generator. Furthermore, property (1) is a "two-step" property in that the consumer information must first reach the controller, and only then be retransmitted to the generator.

In our experiments to test resilience to message loss, we vary the probability of message reception for the PC. Whenever there is a control decision the probability that the PC will receive a message (indicating there was no message loss) can be 0.9, 9.95, 0.97, 0.98, 0.99 and 1.00. To test these properties we use Bayesian interval estimation [22], which is a more expressive variant of the algorithm described in Section 5. This algorithm returns a confidence interval where the probability that the properties are satisfied lie. We can specify the size of the interval, as well

as the confidence coefficient, allowing it to be used for cursory and in-depth analyses. Tables 2 and 3 summarise the results for intervals of 0.04 and a confidence coefficient of 0.95.

| | 1.00 | 0.99 | 0.98 |
|---|---|---|---|
| Prob. | [0.89, 0.93] | [0.87, 0.91] | [0.86, 0.90] |
| # correct/total | 508/557 | 582/651 | 634/720 |
| | 0.97 | 0.95 | 0.9 |
| Prob. | [0.83, 0.86] | [0.75, 0.79] | [0.28, 0.32] |
| # correct/total | 745/879 | 914/1180 | 431/1423 |

**Table 2:** Interval estimation results for property (1)

| | 1.00 | 0.99 | 0.98 |
|---|---|---|---|
| Prob. | [0.95, 0.99] | [0.91, 0.95] | [0.86, 0.90] |
| # correct/total | 180/183 | 399/426 | 608/685 |
| | 0.97 | 0.95 | 0.9 |
| Prob. | [0.82, 0.86] | [0.66, 0.70] | [0.16, 0.20] |
| # correct/total | 754/893 | 998/1461 | 169/971 |

**Table 3:** Interval estimation results for property (2)

As one would expect, a higher probability of message delivery errors will exponentially decrease the probability that the Grid is "safe" by making the generator output deviate too far from what the actual consumption is. We could now focus further studies by restricting message delivery probabilities to 0.97 or 0.98 and 1.00. This analysis could help companies and utilities decide whether to invest in more reliable communication infrastructures depending on how much they perceive the risk to be. This can be further analysed by inspecting the traces that failed to satisfy the desirable properties. Since the cost of increased reliability at the Grid scale is extremely significant, these types of studies quantify risk and allow companies to justify or refute enormous investments in, say, 0.99 or 0.995 reliable infrastructures.

We also see that the stronger property (1) holds less often than the weaker (2), as we foresaw. Furthermore, the discrepancy is directly proportional to the delivery rate. This tells us that communication is central in the Smart Grid. Property (1), by requiring communication from consumers to the PC to the generator, is clearly affected by compounded delays of two hops, while (2) only requires one. It is also strange that this role reverses with low enough delivery probability. This could be

because message loss causes the Grid to adapt more slowly, and thus cope better with fast, but small, variations in power demand.

Network bandwidth is another very configurable network parameter that greatly affects deployment costs. The Grid industry is still deploying networks that send a few thousand bits *per day*. Using the above model with 0.98 message delivery probability but *doubling* the consumer feedback interval from 5 to 10 minutes, we obtain an interval of $[0.80, 0.84]$ for property (1) and $[0.78, 0.82]$ for (2). Once again, a very quick analysis can be performed on this parameter, pruning non-relevant values.

The ability to simulate the system and see its behaviour graphically made it clear that the system is prone to over-production during the evening when the consumers are shutting down. It also has some trouble keeping up with demand in the morning, but to a lesser extent. This indicates that even simple forms of load prediction would greatly help the PC in keeping the system within the safety constraints. Once again, this would be easily implementable within the existing framework.

The ease with which one can change system parameters, specify properties and check them enables us to quickly identify and study the critical parameters in the Smart Grid infrastructure. For more complex scenarios with multiple generators, for instance, the PC's policy regarding which generators to spool up or down will make or break Grid adaptability. In this case, multiple algorithms may quickly be implemented and tested before one is chosen for deployment. This gives us much higher confidence without having to use untested algorithms on very expensive, real-world testing infrastructure.

# 7   Conclusions

In order to check for desirable properties of Smart Grid technologies, we defined Distributed Probabilistic-Control Hybrid Automata as a model for hybrid systems with a dynamic number of probabilistic elements, and Quantified Bounded Linear Temporal Logic to specify properties about a distributed system. We also showed that Bayesian statistical model checking techniques are applicable in this context for verifying QBLTL properties. Finally, we developed a Smart Grid case study where even a preliminary study revealed important cost-benefit relations relevant to full-scale deployment.

# References

[1] 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. LNCS, vol. 736. Springer (1992)

[2] Clarke, E.M., Donzé, A., Legay, A.: Statistical model checking of mixed-analog circuits with an application to a third order delta-sigma modulator. In: Haifa Verification Conference. pp. 149–163 (2008)

[3] Clarke, E.M., Donzé, A., Legay, A.: On simulation-based probabilistic model checking of mixed-analog circuits. Formal Methods in System Design 36(2), 97–113 (2010)

[4] Clarke, E.M., Faeder, J.R., Langmead, C.J., Harris, L.A., Jha, S.K., Legay, A.: Statistical model checking in BioLab: Applications to the automated analysis of T-cell receptor signaling pathway. In: CMSB. pp. 231–250 (2008)

[5] Demongodin, I., Koussoulas, N.: Differential Petri nets: representing continuous systems in a discrete-event world. IEEE Transactions on Automatic Control 43(4), 573–579 (1998)

[6] Henzinger, T.A.: The theory of hybrid automata. In: LICS (1996)

[7] Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to model checking biological systems. In: Degano, P., Gorrieri, R. (eds.) CMSB. LNCS, vol. 5688. Springer (2009)

[8] Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G.J., Rosu, G., Sokolsky, O., Tillmann, N. (eds.) RV. LNCS, vol. 6418. Springer (2010)

[9] Lynch, N.A.: Input/Output automata: Basic, timed, hybrid, probabilistic, dynamic, ... In: Amadio, R.M., Lugiez, D. (eds.) CONCUR. LNCS, vol. 2761, pp. 187–188. Springer (2003)

[10] Lynch, N.A., Segala, R., Vaandrager, F.W., Weinberg, H.B.: Hybrid I/O automata. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) Hybrid Systems. LNCS, vol. 1066. Springer (1995)

[11] Martins, J., Platzer, A., Leite, J.: Statistical model checking for distributed probabilistic-control hybrid automata in the smart grid. Tech. Rep. CMU-CS-11-119, Computer Science Department, Carnegie Mellon University (2011)

[12] Martins, J., Platzer, A., Leite, J.: Statistical model checking for distributed probabilistic-control hybrid automata with Smart Grid applications. In: Qin, S., Qiu, Z. (eds.) ICFEM. LNCS, vol. 6991, pp. 131–146. Springer (2011)

[13] Meseguer, J., Sharykin, R.: Specification and analysis of distributed object-based stochastic hybrid systems. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC. LNCS, vol. 3927. Springer (2006)

[14] Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. 41(2), 143–189 (2008)

[15] Platzer, A.: Quantified differential dynamic logic for distributed hybrid systems. In: Dawar, A., Veith, H. (eds.) CSL. LNCS, vol. 6247. Springer (2010)

[16] Platzer, A.: Stochastic differential dynamic logic for stochastic hybrid programs. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE. LNCS, vol. 6803, pp. 431–445. Springer (2011)

[17] Trivedi, K.S., Kulkarni, V.G.: FSPNs: Fluid stochastic Petri nets. In: Marsan, M.A. (ed.) Application and Theory of Petri Nets. LNCS, vol. 691. Springer (1993)

[18] Wang, Y.C., Komuravelli, A., Zuliani, P., Clarke, E.M.: Analog circuit verification by statistical model checking. In: ASP-DAC. pp. 1–6. IEEE (2011)

[19] Yahav, E., Reps, T., Sagiv, M.: LTL model checking for systems with unbounded number of dynamically created threads and objects. Tech. Rep. TR-1424, Computer Sciences Department, University of Wisconsin (2001)

[20] Younes, H.L.S., Clarke, E.M., Zuliani, P.: Statistical verification of probabilistic properties with unbounded until. In: Davies, J., Silva, L., da Silva Simão, A. (eds.) SBMF. LNCS, vol. 6527. Springer (2010)

[21] Younes, H.L.S., Simmons, R.G.: Statistical probabilistic model checking with a focus on time-bounded properties. Inf. Comput. 204(9), 1368–1409 (2006)

[22] Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to Simulink/Stateflow verification. In: Johansson, K.H., Yi, W. (eds.) HSCC. pp. 243–252. ACM (2010)