POWVER

Technical Report 2020-16

Title: **Sampling Distributed Schedulers for Resilient Space Communication**

Authors: Pedro R. D'Argenio, Juan A.Fraire, Arnd Hartmanns

Report Number: 2020-16

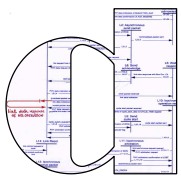
ERC Project: Power to the People. Verified.

ERC Project ID: 695614

Funded Under: H2020-EU.1.1. – EXCELLENT SCIENCE

Host Institution: Universität des Saarlandes, Dependable Systems and Software
Saarland Informatics Campus

Published In: NFM 2020



POWER TO THE PEOPLE.
VERIFIED.
erc

Sampling Distributed Schedulers for Resilient Space Communication

Pedro R. D'Argenio[1], Juan A. Fraire[1], and Arnd Hartmanns[2]

[1] CONICET, Córdoba, Argentina
Saarland University, Saarbrücken, Germany
Universidad Nacional de Córdoba, Córdoba, Argentina
[2] University of Twente, Enschede, The Netherlands

Abstract. We consider routing in delay-tolerant networks like satellite constellations with known but intermittent contacts, random message loss, and resource-constrained nodes. Using a Markov decision process model, we seek a forwarding strategy that maximises the probability of delivering a message given a bound on the network-wide number of message copies. Standard probabilistic model checking would compute strategies that use global information, which are not implementable since nodes can only act on local data. In this paper, we propose notions of distributed schedulers and good-for-distributed-scheduling models to formally describe an implementable and practically desirable class of strategies. The schedulers consist of one sub-scheduler per node whose input is limited to local information; good models additionally render the ordering of independent steps irrelevant. We adapt the lightweight scheduler sampling technique in statistical model checking to work for distributed schedulers and evaluate the approach, implemented in the Modest Toolset, on a realistic satellite constellation and contact plan.

1 Introduction

There is an increasing commercial and scientific interest in deploying large-scale satellite networks in low-Earth orbit (LEO) to collect and distribute information [**?**]. Real-time access to data is, however, only feasible when many satellites align to form a chain of links between a (remote) destination source or destination and a ground station. This vision favours mega-constellations; e.g. SpaceX's Starlink is composed of 12,000 satellites [**?**]. 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—a contact—appears. This gives rise to a *delay-tolerant network* (DTN) [**?**]. Originally intended for interplanetary networks [**?**], the DTN architecture has been identified as a disruptive approach for LEO constellations which allows for a better utilisation of communication opportunities [**?**]. In DTN, there is no upper bound on the propagation delay, and no expectation of continuous or bi-directional end-to-end connectivity. While DTN satellite constellations do not work for e.g. voice

services, they can network Earth observation and high-latency data service missions. In particular, they can utilise slower inter-satellite links by advance forwarding [**?**], adapt transmission schedules to limited battery conditions [**?**], work with constrained spacecraft antennas and subsystem architectures [**?**], and tailor communication resources to fit mission traffic and routes [**?**, **?**]. Ultimately, the DTN approach enables sparse topologies of fewer and and cheaper satellites [**?**].

A *contact* is the opportunity to establish a temporal communication link. As a DTN node's network state information may be inaccurate or obsolete, traditional Internet routing schemes cannot be used. Space DTN routing approaches (both near-Earth and deep-space) thus seek to exploit the *a priori* knowledge of contacts: inter-satellite and satellite-to-ground contacts can be precomputed based on the orbital elements and communication parameters. The result is a *contact plan* [**?**]. We visualise a simple plan for four satellites—nodes N_1 through N_4—in Fig. 1. A bent vertical arrow indicates a contact from the arrow's origin to its target node. We abstract real time into discrete time slots T_1 through T_5; actual contact plans would show actual time intervals of varying durations (and potentially overlapping) with up to sub-second precision. Contact plans describe the expected network connectivity over time. They are the input for centralised or distributed DTN routing procedures. Existing solutions turn contact plans into e.g. time-expanded graphs [**?**] or contact graphs [**?**] on which routing calculations can be performed efficiently. Contact graphs have notably been validated by technological demonstrations in orbit [**?**, **?**].

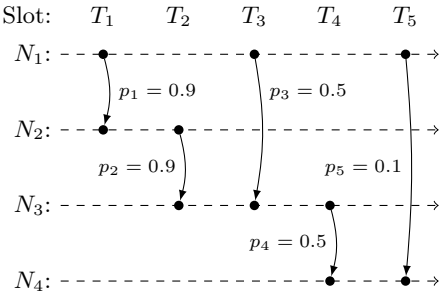


Fig. 1. Abstract uncertain contact plan

In practice, the actual contacts may differ from the original plan due to failures or incomplete/inaccurate knowledge at the time the plan was computed. Space DTN in particular face fault-prone nodes [**?**], interference-sensitive communication links [**?**], and inaccurate orbit determination and station keeping procedures [**?**]. We thus need *uncertain contact plans* where contacts may fail for various reasons. Based on statistical data, we can annotate every contact with its success probability. In Fig. 1, we use probabilities p_1 through p_5 for illustration. Given an uncertain contact plan, we would then like to find a routing strategy that maximises the probability that the message is delivered to its destination. To increase that probability, we can allow *copies* of the message that propagate along different paths. However, as typical DTN satellites have limited resources, we also want to bound the number of copies. Existing routing schemes only perform well under perfectly known or fully unknown contact plans [**?**, **?**], leaving significant room for improvement for uncertain contact plan routing.

The routing problem in uncertain contact graphs with bounded copies matches very well with the modelling capabilities of Markov decision processes (MDP) [**?**]. They combine discrete probabilistic choices as in discrete-time Markov chains,

which can represent contact failures, with nondeterministic choices as in Kripke structures [?], which can represent the routing options. Given an MDP model, we can use probabilistic model checking (PMC) [?] to determine the routing strategy (corresponding to the *scheduler* in PMC) with the highest probability for eventual message delivery. Raverta et al. [?] recently used this approach with optimisations that exploit the structure of the DTN routing models. However, PMC computes *global-information* schedulers, which take the local states of *all* nodes into account to make the optimal decision; they are thus *unimplementable*.

Example 1. In the contact plan of Fig. 1, assume we can send one message per slot. The highest-probability route from N_1 to N_4 is N_1-N_2-N_3-N_4 (in slots T_1, T_2, and T_4, with probability 0.405). The second-best is N_1-N_3-N_4 (in T_3 and T_4, $p = 0.25$). Sending directly from N_1 to N_4 is least reliable. If N_1 starts with two copies of the message, then the first should be sent to N_2 in T_1. In slot T_3, N_1 can then either try to send the remaining copy to N_3, or keep it for slot T_5. We will show in Sect. 2 that the best choice for node N_1 computed by PMC is to send in T_3 iff node N_3 does not already have a copy of the message, i.e. if communication in slot T_1 or T_2 failed. In a space DTN, N_1 *cannot know this!*

PMC is thus not well suited for space DTN routing. The underlying problem of applying PMC to distributed systems was recognised almost 20 years ago [?,?,?], and led to the development of the notion of *(strongly) distributed schedulers* [?,?] that only act on locally observable information. However, depending on the exact formalism and definition used, PMC for these schedulers is undecidable [?] in general, and NP-hard in the memoryless case [?, ?].

In this paper, we propose to use statistical model checking (SMC) [?, ?] with lightweight scheduler sampling (LSS) [?] in place of PMC to obtain routing strategies with a high probability for message delivery. Our contributions are (1) a modern and practical definition of distributed schedulers and models (in Sect. 3) appropriate for the space DTN setting that matches the compositional state-based modelling approach with undirected synchronisation common to today's probabilistic modelling languages [?, ?, ?, ?] and tools [?, ?, ?, ?]; (2) an adaptation and implementation of SMC with LSS for distributed schedulers (in Sect. 4), and (3) a modelling pattern and SMC-based analysis toolchain for routing in space DTN with uncertain contact plans (in Sect. 5). We start (in Sect. 2) with a simplified but complete definition of the compositional MDP formalism that underpins our approach, introducing a detailed model for Fig. 1 along the way. We end (in Sect. 5) with an experimental evaluation of our new technique on a realistic satellite constellation model and contact plan. A package with the tools and models needed to replicate our experiments is available at modestchecker.net/nfm2020/artifact.zip; accesses to this URL are not logged.

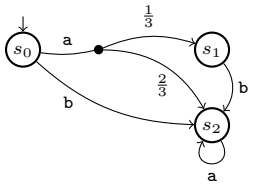
2 Scheduling in Markov Decision Processes

Preliminaries. $\mathbb{Z}$ is the set of integer numbers. We write $[a, b]$ for the real interval $\{\, x \in \mathbb{R} \mid a \leq x \leq b \,\}$. Given a set S, its powerset is 2^S. A proba-

bility distribution over S is a function $\mu\colon S \to [0,1]$ with countable support $spt(\mu) \stackrel{\text{def}}{=} \{\, s \in S \mid \mu(s) > 0 \,\}$ and $\sum_{s \in spt(\mu)} \mu(s) = 1$. $Dist(S)$ is the set of all probability distributions over S. We write $\{\, x_1 \mapsto y_1, \dots \,\}$ for the function that maps each x_i to y_i, and if necessary implicitly maps to 0 all other x. Thus we can e.g. write $\{\, s \mapsto 1 \,\}$ for the *Dirac distribution* that assigns probability 1 to s. For a tuple $t = \langle y_1, \dots, y_n \rangle$, $t[i] \stackrel{\text{def}}{=} y_i$ is its i-th component.

Definition 1. *A* Markov decision process *(MDP) is a tuple $M = \langle S, s_I, A, T \rangle$ where S is a finite set of* states *with* initial state $s_I \in S$*, A is a finite set of* actions*, and $T\colon S \to 2^{A \times Dist(S)}$ is the* transition *function with $T(s) \neq \varnothing$ for all $s \in S$. The set of all* transitions *is $Tr(M) \stackrel{\text{def}}{=} \cup_{s \in S} T(s)$; it must be finite.*

We also write $s \xrightarrow{a}_T \mu$ for $\langle a, \mu \rangle \in T(s)$, and may omit the $_T$ subscript. An element of $spt(\mu)$ is a *branch* of transition $s \xrightarrow{a}_T \mu$. To leave a state, we first choose a transition, then select the next state probabilistically among its branches. An MDP with $\forall s\colon |T(s)| = 1$ is a *discrete-time Markov chain* (DTMC). We draw MDP as shown above on the right: this MDP has three states with $s_I = s_0$, four transitions, and five branches. For transitions with a single branch, we omit the dot and probability 1. We have $s_0 \xrightarrow{\mathtt{a}} \{\, s_1 \mapsto \frac{1}{3}, s_2 \mapsto \frac{2}{3} \,\}$.



Modelling with MDP directly is cumbersome; we instead use a higher-level modelling language like MODEST [?,?] that extends MDP with discrete variables and parallel composition. Given a set of (integer-valued) *variables* X, let $Val_X \stackrel{\text{def}}{=} X \to \mathbb{Z}$ be their *valuations*. Let Bxp_X and Ixp_X contain all Boolean and integer *expressions* over the variables in X, respectively. We omit $_X$ subscripts if clear from the context. For $e \in Bxp$ ($e \in Ixp$), let $v(e) \in \{\, true, false \,\}$ ($v(e) \in \mathbb{Z}$) be the value of e in $v \in Val$. Finally, let $Upd \stackrel{\text{def}}{=} X \mapsto Ixp$ be the set of *updates* that map each variable to an expression determining the value assigned to it.

Definition 2. *An* MDP with variables *(VMDP) is a tuple $M = \langle Loc, \ell_I, A, X, x_I, E \rangle$ where Loc is a set of* locations *with* initial location $\ell_I \in Loc$*, A is a set of* actions*, X is a set of variables with initial values given by $x_I \in Val$, and $E\colon Loc \to 2^{Bxp \times A \times Dist(Upd \times Loc)}$ is the* edge *function. All sets must be finite.*

We write $s \xrightarrow{g,a}_E \nu$ for $\langle g, a, \nu \rangle \in E(\ell)$, and may omit the $_E$ subscript. An edge in a VMDP has a *guard* g that determines whether the edge is *enabled*. A branch of an edge carries an update u that changes the variables' values. Formally:

Definition 3. *Given a VMDP $M = \langle Loc, \ell_I, A, X, x_I, E \rangle$, its* semantics *is the MDP $[\![M]\!] \stackrel{\text{def}}{=} \langle Loc \times Val, \langle \ell_I, x_I \rangle, A, T \rangle$ with T the smallest function satisfying*

$$\frac{\ell \xrightarrow{g,a}_E \nu \wedge v(g)}{\langle \ell, v \rangle \xrightarrow{a}_T \{\, \langle \ell', v' \rangle \mapsto \sum_{\{u \mid u \in Upd \wedge v' = \{x \mapsto v(u(x))\}\}} \nu(\langle u, \ell' \rangle) \mid \ell' \in Loc, v' \in Val \,\}}$$

We must restrict to VMDP whose semantics is finite and deadlock-free.

Example 2. Fig. 2 shows four VMDP N_1 through N_4 that model the nodes of Fig. 1. Every node has a variable c_i to track the number of message copies it

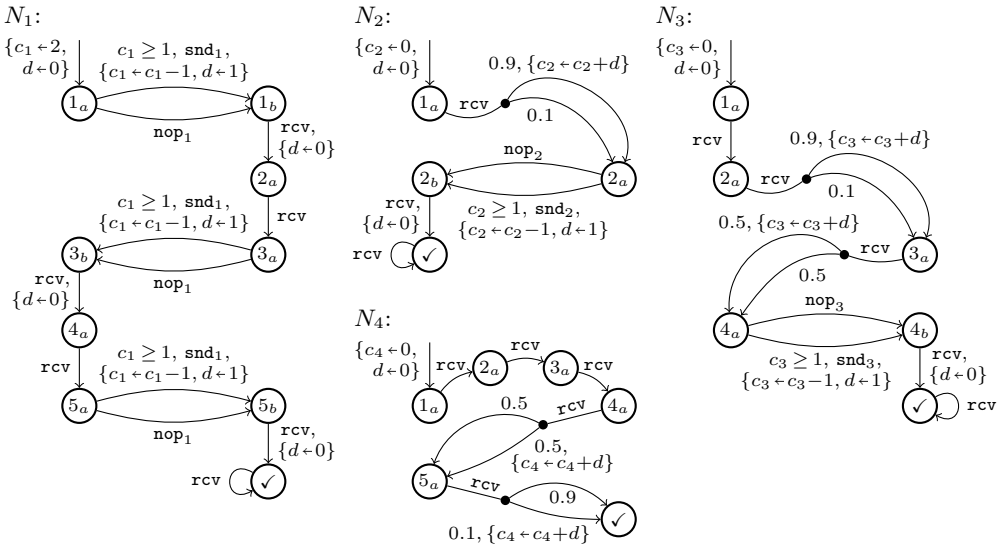


Fig. 2. Four VMDP modelling the nodes of the example contact plan

owns. We write $x \leftarrow e$ for the mapping of variable x to value or expression e. In every slot where a node N_i *can* send, it has a choice between two transitions labelled $\texttt{nop}_i$ (do not send) and $\texttt{snd}_i$ (send one copy: decrement c_i, set d to 1). In a slot T_j where N_i can receive, it always tries to do so via action `rcv`; this succeeds with probability p_j as given in Fig. 1. If the sender decided not to send, then a successful receive has no effect on c_i because d is zero. The *parallel composition* of these four VMDP models the entire contact plan, with the nodes synchronising on shared action `rcv` and exchanging data via shared variable d.

Definition 4. *Given two VMDP* $M_i = \langle Loc_i, \ell_{I_i}, A_i, X_i, x_{I_i}, E_i \rangle$, $i \in \{1, 2\}$, *a finite set* A *of actions, and a* synchronisation relation

$$sync \subseteq (A_1 \uplus \{\bot\}) \times (A_2 \uplus \{\bot\}) \times A,$$

their parallel composition *is*

$$M_1 \parallel_{sync} M_2 \stackrel{\text{def}}{=} \langle Loc_1 \times Loc_2, \langle \ell_{I_1}, \ell_{I_2} \rangle, A, X_1 \cup X_2, x_{I_1} \cup x_{I_2}, E \rangle$$

where E *is the smallest function that satisfies the inference rules*

$$\frac{\ell_1 \xrightarrow{g_1, a_1}_{E_1} \nu_1 \qquad \langle a_1, \bot, a \rangle \in sync}{\langle \ell_1, \ell_2 \rangle \xrightarrow{g_1, a}_{E} \{\, \langle \langle \ell_1', \ell_2 \rangle, u_1 \rangle \mapsto \nu_1(\langle \ell_1', u_1 \rangle) \mid \langle \ell_1', u_1 \rangle \in spt(\nu_1) \,\}} \ (ind_1),$$

$$\frac{\ell_1 \xrightarrow{g_1, a_1}_{E_1} \nu_1 \qquad \ell_2 \xrightarrow{g_2, a_2}_{E_2} \nu_2 \qquad \langle a_1, a_2, a \rangle \in sync}{\langle \ell_1, \ell_2 \rangle \xrightarrow{g_1 \wedge g_2, a}_{E} \{\, \langle \langle \ell_1', \ell_2' \rangle, u_1 \cup u_2 \rangle \mapsto \nu_1(\langle \ell_1', u_1 \rangle) \cdot \nu_2(\langle \ell_2', u_2 \rangle) \mid \dots \,\}} \ (syn),$$

plus a rule ind_2 *for* M_2 *that is symmetric to* ind_1*, with* … *in syn replaced by*

$$\langle \ell_1', u_1 \rangle \in spt(\nu_1) \wedge \langle \ell_2', u_2 \rangle \in spt(\nu_2) \wedge u_1 \cup u_2 \text{ is a function.}$$

Inference rules ind_1 and ind_2 allow the individual VMDP to proceed independently if allowed by *sync*; rule *syn* covers the case where they synchronise on a

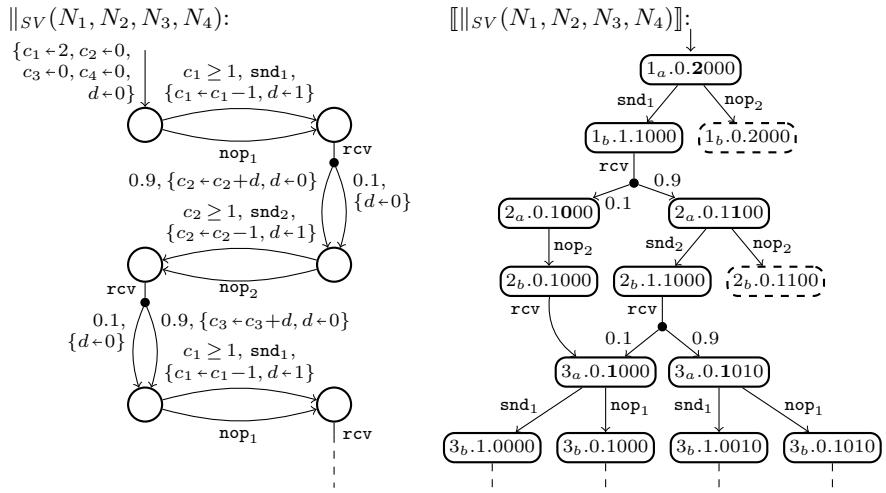


Fig. 3. Excerpt of the network of the node VMDP (left) and its semantics (right)

pair of actions. An element of *sync* is called a *synchronisation vector*; we also write $\langle a_1, a_2 \rangle \mapsto a$ for vector $\langle a_1, a_2, a \rangle$. This flexible form of parallel composition can be generalised to more than two VMDP with longer synchronisation vectors. We refer to such a general parallel composition as a *network* of VMDP and write e.g. $\|_{SV}(M_1, M_2, M_3)$ for the network of M_1, M_2, and M_3, with the set of synchronisation vectors SV. The case in *syn* where $\nu_1(\langle \ell_1', u_1 \rangle) \cdot \nu_2(\langle \ell_2', u_2 \rangle) \neq 0$ but u_1 and u_2 assign different expressions to a shared variable (s.t. $u_1 \cup u_2$ is not a function) is a modelling error, and we do not consider networks of VMDP with such inconsistent assignments, or with inconsistent initial valuations x_{I_1} and x_{I_2}.

Example 3. Fig. 3 shows an initial fragment of the network of the node VMDP N_1 through N_4 on the left. We omit location labels. Synchronisation vectors are

$$SV = \{\, \langle \mathtt{rcv}, \mathtt{rcv}, \mathtt{rcv}, \mathtt{rcv} \rangle \mapsto \mathtt{rcv}, \langle \mathtt{snd}_1, \bot, \bot, \bot \rangle \mapsto \mathtt{snd}_1, \dots \}:$$

In every slot, the nodes first independently make their choices of whether to send or not ($\mathtt{snd}_i$ vs. $\mathtt{nop}_i$); then they synchronise on $\mathtt{rcv}$ to perform their receive actions simultaneously. Updates are atomic, i.e. all right-hand sides are evaluated first, thus there is no order dependency in e.g. $\{c_3 \leftarrow c_3 + d, d \leftarrow 0\}$. On the right of Fig. 3, we show a fragment of the network's MDP semantics. We write $\ell.v_d.v_{c_1}v_{c_2}v_{c_3}v_{c_4}$ for state $\langle \langle \dots, \ell, \dots \rangle, \{\, d \mapsto v_d, c_1 \mapsto v_{c_1}, \dots \} \rangle$.

A *path* defines the behaviour of an MDP by resolving all nondeterminism and probabilistic choices. A *scheduler* resolves the nondeterminism only.

Definition 5. *Let M be an MDP as in Def. 1. A* path *π of M is an infinite sequence $\pi = s_0\, tr_0\, s_1 \dots \in (S \times Tr(M))^\omega$ such that, for all $i \in \{0, \dots\}$, we have $tr_i = \langle a, \mu \rangle \in T(s_i)$ and $\mu(s_{i+1}) > 0$. $\Pi(M)$ is the set of all paths of M. We write $\Pi_{fin}(M)$ for the set of all* path prefixes *π_{fin} ending in a state. The last state of π_{fin} is denoted $last(\pi_{fin})$.*

Definition 6. *Let M be an MDP as above. A (randomised, history-dependent)* scheduler *is a function $\sigma\colon \Pi_{fin}(M) \to Dist(Tr(M))$ such that $\forall s \in S\colon \sigma(s) = tr \Rightarrow tr \in T(s)$. We write $\mathfrak{S}(M)$ for the set of all schedulers of M. A* deterministic *scheduler is in $\Pi_{fin}(M) \to Tr(M)$; a (deterministic)* memoryless *scheduler is in $S \to Tr(M)$. A memoryless scheduler σ_{ml} defines a corresponding deterministic scheduler σ_{det} by $\sigma_{det}(\pi_{fin}) = \sigma_{ml}(last(\pi_{fin}))$ and a deterministic scheduler σ_{det} defines a scheduler σ by $\sigma(\pi_{fin}) = \{\, \sigma_{det}(\pi_{fin}) \mapsto 1 \,\}$.*

If we "apply" a scheduler σ to an MDP M, it removes all nondeterminism and we obtain a DTMC $M|_\sigma$, whose paths can be measured and assigned probabilities according to the transitions' distributions. Formally, these probability measures over sets of paths are built via cylinder sets; we refer the interested reader to e.g. [**?**] for details. Given a set of goal states $G \subseteq S$ and a scheduler σ, we are interested in the probability of the measurable subset of paths of $\Pi(M|_\sigma)$ including a state in G. We call the supremum (infimum) when ranging over all schedulers $\sigma \in \mathfrak{S}(M)$ the *maximum* (*minimum*) *reachability probability*. There is always a memoryless scheduler that achieves the supremum respectively infimum [**?**].

Example 4. For our example contact plan, we are interested in the maximum probability to reach a state where N_4 has at least one copy, and the corresponding scheduler. Expanding the MDP of Example 3, we can calculate that probability to be 0.493. The optimal choices are to send in slots T_1 and T_2; then in slot T_3,

- if we are in state $3_a.0.1000$ (i.e. if the first copy was lost on the way from N_1 via N_2 to N_3): send from N_1 to N_3, and send from N_3 to N_4 in T_4;
- if we are in state $3_a.0.1010$ (i.e. the first copy made it to N_3), do not send from N_1 to N_3 (since N_3 already has a copy and can only send to N_4 once), then send in both T_4 and T_5.

Thus the optimal choice of node N_1 in slot T_3 depends on whether node N_3 has a copy of the message or not. In a real distributed setting, N_1 cannot know this.

3 Distributed Scheduling

Example 4 showed that a scheduler that maximises the probability for eventual message delivery may need complete *global* information. In a distributed system like a satellite constellation, such a scheduler cannot be implemented; the satellites must decide whether to send based on their local state (here: their number of message copies c_i) only. This problem was initially studied with a focus on aspects of compositionality [**?**], ignoring algorithms except for a simple partial-information setting [**?**]. Giro et al. [**?**] defined *distributed schedulers*, for which computing and approximating optimal probabilities is in general impossible [**?**]. The formalisms of [**?**] and [**?**] do not provide for scheduling the interleaving of parallel components, i.e. deciding which component acts first in case both of them have an enabled edge in the same state. This gap was filled in [**?**] along with the introduction of *strongly distributed* schedulers. Though model checking remains undecidable in general, [**?**] proved that for memoryless schedulers it is "only" NP-hard. Consequently, no model checker as of today supports (strongly)

distributed schedulers. The only prototype [?] was restricted to time-bounded reachability and suffered from exponential explosion in intermediate model sizes. Other prototype tools only provide overapproximations [?, ?]. Our formalism of Sect. 2 is more expressive than those previously considered by allowing interleaving and information exchange via both synchronisation and (shared) variables.

In this paper, we restrict to a memoryless deterministic variant of distributed schedulers adapted to our formalism (Sect. 3.1), and we define a desirable characteristic of models that makes scheduling decisions about interleavings irrelevant (Sect. 3.2). We prefer memoryless distributed schedulers because history-dependent ones need infinite memory (which is again unimplementable), and using randomised schedulers would add additional unpredictability to the behaviour of the system, which is undesirable from the practitioner's point of view.

3.1 Simple Distributed Schedulers

Definition 7. *Given a network of VMDP* $M = \|_{SV}(M_1, \ldots, M_n)$ *and* $i \in \{1, \ldots, n\}$*, let* $read(M_i)$ *be the set of all variables that occur in the guards of edges of* M_i *or on the right-hand sides of assignments in the updates in* M_i*. A state in* $[\![M]\!] = \langle S, s_I, A, T\rangle$ *has the form* $s = \langle\langle \ell_i, \ldots, \ell_n\rangle, v\rangle$ *where* ℓ_i *is the current location of* M_i *and* $v \in Val_{\cup_i X_i}$*. Then the* M_i-projection *of* s *is* $s{\downarrow_{M_i}} \stackrel{\text{def}}{=} \langle \ell_i, v{\downarrow_{read(M_i)}}\rangle$ *with* $v{\downarrow_{read(M_i)}} \stackrel{\text{def}}{=} \{\, x \mapsto v(x) \mid x \in read(M_i) \,\} \in Val_{read(M_i)}$*. Let* $S{\downarrow_{M_i}} \stackrel{\text{def}}{=} \{\, s{\downarrow_{M_i}} \mid s \in S \,\}$ *be the set of all projected states. Every transition* $tr = s \xrightarrow{a} s'$ *in* $[\![M]\!]$ *can be traced back to a unique (generalised) synchronisation vector* $sv \in SV$ *through the rules of defs. 3 and 4. We say that* M_i *is* involved in *tr if* $sv[i] \neq \bot$*. We write* $I_t(M_i)$ *for the set of all transitions* M_i *is involved in. For a transition* tr*,* $I_c(tr) \stackrel{\text{def}}{=} \{\, M_i \mid tr \in I_t(M_i) \,\}$ *is the set of all components involved in* tr*, and for a set of transitions* TR*,* $I_c(TR) \stackrel{\text{def}}{=} \bigcup_{tr \in TR} I_c(tr)$*.*

Simple distributed schedulers now consist of an interleaving scheduler (to select the component to perform the next transition) plus a local scheduler per component that only sees the component's projection of the current state:

Definition 8. *A* simple distributed scheduler *for* M *as above is a tuple* $\sigma_{sd} = \langle \sigma_I, \sigma_1, \ldots, \sigma_n\rangle$ *of an* interleaving scheduler $\sigma_I \colon S \to \mathbb{N}$ *and* n local schedulers $\sigma_i \colon S \to I_t(M_i) \cup \{\, \bot \,\}$ *for* $i \in \{1, \ldots, n\}$ *s.t.* $\sigma_I(s) \in \{\, i \mid T(s) \cap I_t(M_i) \neq \emptyset \,\}$*,* $\sigma_{\sigma_I(s)}(s) \in T(s) \cap I_t(M_i)$*, and* $s{\downarrow_{M_i}} = s'{\downarrow_{M_i}} \Rightarrow \sigma_i(s) = \sigma_i(s')$ *for all* $s, s' \in S$*,* $i \in \{1, \ldots n\}$*. It defines a memoryless scheduler* σ *for* $[\![M]\!]$ *by* $\sigma(s) = \sigma_{\sigma_I(s)}(s)$*.*

Simple distributed schedulers differ from the partial-information setting of [?] by combining multiple projections in one scheduler. They also differ from the (strongly) distributed schedulers of [?, ?] by hiding information from states but not admitting information disclosure via synchronisation at all (by virtue of being memoryless). As such, they match the nowadays standard state-based approach to probabilistic verification where transition labels are only used to determine synchronisations, as embedded in e.g. the Jani [?], Modest [?, ?], and Prism modelling languages [?], and implemented in probabilistic model checkers like Epmc [?], the Modest Toolset [?], Prism [?], and Storm [?].

Unlike [?,?], our formalism does not partition actions into inputs and outputs; thus every component involved in one of our (undirected) transitions can be chosen by the interleaving scheduler. Interleaving schedulers are problematic: They may disclose global information by scheduling transitions in certain orders (see [?]), the nondeterminism they deal with is in fact uncontrollable, and they would again be unimplementable in fully distributed systems.

3.2 Good-for-Distribution Models

In [?], the problem of information disclosure by the interleaving scheduler was solved by restricting to *strongly* distributed schedulers which do not reveal information in this way *by definition*. Since this is not a constructive approach, and since we cannot implement an interleaving scheduler anyway, we only create models where the interleaving scheduler is by construction irrelevant.

Definition 9. *A network of VMDP* $M = \|_{SV}(M_1, \ldots, M_n)$ *is* good for distributed scheduling *w.r.t. reachability of goal set* G *if in all states* $s \in S$ *of* $[\![M]\!] = \langle S, s_I, A, T\rangle$ *where* $|T(s)| > 1 \wedge |\{\, i \mid T(s) \cap I(M_i) \neq 0 \,\}| > 1$ *we have*

$$\forall\, s \xrightarrow{a} s' \colon s \in G \Leftrightarrow s' \in G \;\vee\; \forall\, s \xrightarrow{a} s' \colon s \in G \Leftrightarrow s' \notin G, \tag{1}$$

$$\forall\, i \in \{\, 1, \ldots, n \,\} \colon\; |I_t(M_i) \cap T(s)| > 1 \;\Rightarrow\; I_c(I_t(M_i) \cap T(s)) = \{\, M_i \,\}, \tag{2}$$

and

$$s \xrightarrow{a} s' \;\Rightarrow\; \forall\, M_c \in \{\, M_1, \ldots, M_n \,\} \setminus I_c(s \xrightarrow{a} s') \colon s'{\downarrow_{M_c}} = s'{\downarrow_{M_c}}. \tag{3}$$

In words, a network is good if in all states where the interleaving scheduler has a nontrivial choice among multiple components, (1) it cannot influence whether we directly move to a goal state, (2) no component has a local choice involving at least one synchronising transition, and (3) no transition can change variables that are visible to a component not involved in the transition.

Lemma 1. *For a good-for-distributed-scheduling network of VMDP* M*,* fair *interleaving schedulers* σ_{I_1} *and* σ_{I_2}*, and a set* G *of goal states, the maximum (minimum) probability to reach a state in* G *under* σ_{I_1} *is the same as under* σ_{I_2}*.*

An interleaving scheduler is *fair* if, on every cycle in $[\![M]\!]$, it chooses every available component at least once. This is a reasonable assumption in practice.

Proof idea. In good-for-distributed-schedulers models, the interleaving scheduler cannot cause an unrelated transition to become disabled. The restriction of $[\![M]\!]$ induced by an interleaving scheduler together with condition 1 and the fairness requirement is thus a (stronger) variant of a partial order reduction using ample sets as in [?] (with condition **A5'**). A more detailed argument is in the appendix.

The fairness requirement is trivially satisfied for acyclic models (modulo self-loops in leaf states) like our running example, or if we replace σ_I by σ_{uni}, the randomised scheduler that picks a component uniformly at random every time.

Example 5. Our running example from examples 2 and 3 is good for distributed scheduling: in every state, either a single node internally decides between $\texttt{snd}_i$

and $\texttt{nop}_i$ or all nodes synchronise on `rcv`. In the former states, the interleaving scheduler has no choice, thus no conditions apply. In the latter states, conditions 2 and 3 are directly satisfied. The only way to move into a goal state is from a state of the latter kind, thus condition 1 also holds.

While simple distributed schedulers restrict local schedulers from *reading* certain variables, a good-for-distributed-scheduling model restricts component edges from *writing* to certain variables. A model checker can easily determine whether a model is good, but a precise syntactic check on the network-of-VMDP level is not possible. We can still use syntactic overapproximations, e.g. by requiring that locations with multiple outgoing edges have only internal edges not writing to shared variables, and that a shared variable may only be updated by synchronising edges involving all components that read the variable. Our running example does not satisfy this syntactic restriction due to the writes to d on $\texttt{snd}_i$-labelled edges. We solve this (in Sect. 5.1) by using a feature of Modest and Jani that allows to specify *sequences* of atomic updates, moving the d-write assignments onto the `rcv` transitions and executing them before the d-read assignments.

4 Lightweight Distributed Scheduler Sampling

Since PMC for distributed schedulers is undecidable or computationally infeasible, we propose a different approach to find useful high-probability strategies: we combine statistical model checking (SMC) [**?**,**?**] with a new variant of lightweight scheduler sampling (LSS) [**?**] that samples only simple distributed schedulers. As our models are good for distributed scheduling by construction (see Sect. 5.1), only the satellites' local choices are relevant and we can replace the interleaving scheduler by uniformly random choices.

SMC with LSS. SMC is Monte Carlo simulation with formal models: perform several *simulation runs* using a pseudo-random number generator (PRNG) to resolve probabilistic choices according to the model's distributions, then return a statistical estimate and confidence for the probability of interest. As-is, SMC does not consider the optimisation problem over nondeterminism posed by MDP. LSS is to date the only extension of SMC that takes scheduling into account (in contrast to e.g. Prism or Uppaal smc [**?**], which always use the uniformly random scheduler) and also preserves SMC's constant memory usage (in contrast to learning-based approaches like [**?**]). The basic idea of LSS is as follows:

1. Randomly select m 32-bit integers. Each of them is a *scheduler identifier* σ.
2. For each σ, perform standard SMC under the scheduler identified by σ.
3. Return the maximum (or minimum) result and the corresponding σ.

Due to the multiple tests, we need to adjust SMC's statistical evaluation [**?**, **?**]. Within step 2, when there is a choice between n transitions from state s, LSS concatenates the bit-vector representations of s and σ into $s.\sigma$, hashes the result into a 32-bit number $h = \mathcal{H}(s.\sigma)$, and picks the $h \bmod n$-th transition. $\mathcal{H}$ is deterministic so that σ defines a fixed memoryless scheduler. If $\mathcal{H}$ is also uniform (w.r.t. all bits of $s.\sigma$), then LSS uniformly samples memoryless schedulers.

Input: Network of VMDP $M = \|_{SV}(M_1, \ldots, M_n)$ with $[\![M]\!] = \langle S, s_I, A, T\rangle$, goal set $G \subseteq S$, $\sigma \in \mathbb{Z}_{32}$, $\mathcal{H}$ uniform deterministic, PRNG $\mathcal{U}_{\text{pr}}$.

1 $s := s_I$
2 **while** $s \notin G$ **do** *// break on goal state*
3 **if** $\forall\, s \xrightarrow{a} \mu\ \exists\, s' \in S\colon \mu = \{\, s' \mapsto 1 \,\}$ **then break** *// break on self-loops*
4 $C := \{\, j \mid T(s) \cap I_t(M_j) \neq \emptyset \,\}$ *// get active components*
5 $i := \mathcal{U}_{\text{pr}}(\{\, j \mapsto \frac{1}{|C|} \mid j \in C \,\})$ *// select component uniformly*
6 $T_i := T(s) \cap I_t(M_i)$ *// get component's transitions*
7 $\langle a, \mu\rangle := (\mathcal{H}(\sigma.s{\downarrow_{M_i}}) \bmod |T_i|)$-th element of T_i *// schedule local transition*
8 $s := \mathcal{U}_{\text{pr}}(\mu)$ *// select next state according to μ*
9 **return** $s \in G$

Algorithm 1. Lightweight simple distributed scheduler sampling

The result of SMC with LSS is an *under*approximation (overapprox.) of the maximum (min.) probability. It can thus e.g. disprove safety, or show schedulability—which is what we are interested in. Its efficiency—how large an m we need to get a good approximation—is determined by the probability of sampling a near-optimal scheduler, and thus strongly depends on the model at hand.

LSS for distributed schedulers. In contrast to PMC, LSS is easy to adapt to different classes of schedulers by changing the input to $\mathcal{H}$. In Alg. 1, we show pseudocode for our adaptation to simple deterministic schedulers for MDP. We write $\mathcal{U}(\mu)$ for the pseudo-random selection by PRNG $\mathcal{U}$ of a value from $spt(\mu)$ according to the probabilities of μ. Line 5 implements the interleaving scheduler; we assume good-for-distributed-scheduling models and thus use uniform random resolution here, but could equally replace the line by $i := \mathcal{H}(\sigma.s) \bmod |T_i|$. Line 7 implements the local scheduler, whose input is restricted to the chosen component's projection of the current state. In line 8, we use $\mathcal{U}_{\text{pr}}$ to pseudo-randomly select the successor state according to the distribution determined by the scheduled transition. Line 3 terminates the simulation negatively if we find a state that only has deterministic self-loop transitions; this suffices for our space DTN models, but could be replaced by smarter loop detection or methods like [**?**, **?**].

We have implemented Alg. 1 in MODES [**?**], the statistical model checker of the MODEST TOOLSET [**?**]. MODES is implemented in C#, freely available at modestchecker.net, runs on 64-bit Linux, macOS, and Windows, and is faster than other current general-purpose SMC tools [**?**, Section 7.1]. Its input languages are MODEST [**?**, **?**] and the tool-independent JANI model exchange format [**?**]. It provides both variants of line 5 discussed above, and implements corrected statistical tests as well as two-phase and smart sampling.

5 Scheduling Satellite Communication

To apply our new LSS method of Sect. 4 to space DTN, we created the toolchain shown in Fig. 4. We use the STK tool by AGI [**?**] and the Contact Plan Designer

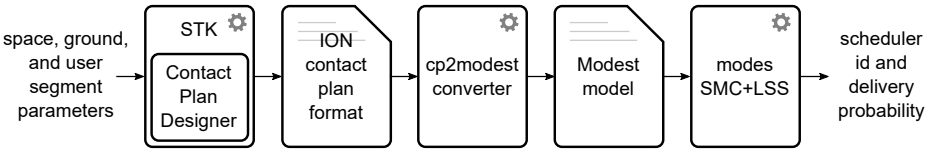


Fig. 4. Satellite DTN routing scheduling toolchain

plugin [**?**] to model the scenario and export the contact plan to a file in Interplanetary Overlay Network format (ION) [**?**]. This plan contains the precise real-time communication windows; we developed the Python CP2MODEST tool that, given such a plan, message source and destinations, and a bound on the number of copies, (1) abstracts the plan into the form of Fig. 1 with discrete non-overlapping slots[3], and (2) creates a MODEST model representing a network of VMDP of the same structure as Example 2. We then run MODES with LSS for simple distributed schedulers to obtain a good scheduler and its probability. Compared to the previous PMC-based work [**?**], we not only generate guaranteed implementable schedules, but also support multiple message copies.

5.1 Modelling Satellite DTN

The model of Fig. 1 given in Example 2 uses unidirectional unreliable communication: for every contact, one node is predetermined as sender; if communication fails, the copy is lost. We also assumed that there is at most one contact and we transmit at most one copy per slot. The models generated by CP2MODEST keep the same structure, but assume bidirectional half-duplex communication, support multiple contacts per slot (including one node having a contact with multiple others; it then needs to choose with whom to communicate), allow sending multiple copies (to one node in one slot), and use either unreliable or acknowledgment-based communication. The latter allows the sender to determine whether a message was successfully received, thus in such models it will keep the sent copies in case of failure. Unreliable communication is a natural choice in deep-space networks, the original application of DTN, while acknowledgment mechanisms are possible and typical in LEO constellations.

The models created by CP2MODEST consist of one *process* definition per node. Listing 1 shows an excerpt of node N_1 in the plan of Fig. 1, but with acknowledgments and all mechanisms to support multiple contacts per slot in place. Like in the VMDP of Example 2, in every slot in which a node has a contact, we have a choice of action followed by a global synchronisation on action `rcv`. Here, in T_1, the choices are to (a) do nothing (line 3), (b) try to send one (line 4) or two copies (line 8) to N_2, or (c) listen to N_2 (line 12). We use distinct actions for every choice to make it easier to trace the best scheduler's decisions later on. Global variable `data1` takes the role of d of Example 2, but only for messages

[3] The abstraction underapproximates contacts (it may only remove or shorten communication opportunities), so a strategy for the abstract plan is always implementable.

```
1  process Node1(int(0..COPIES) copies) {
2     alt {                                          // slot 1: contact with node 2
3     :: nop1; rcv                                     // do nothing in this slot
4     :: when(copies >= 1) snd1to2_1;                  // send one copy to node 2
5        rcv palt {
6        :0.9: {= data1 = 1, dest1 = 2, 2: copies -= ack2==1 ? 1 : 0 =}
7        :0.1: {= /* lost */ =} }
8     :: when(copies >= 2) snd1to2_2;                  // send two copies to node 2
9        rcv palt {
10       :0.9: {= data1 = 2, dest1 = 2, 2: copies -= ack2==1 ? 2 : 0 =}
11       :0.1: {= /* lost */ =} }
12    :: rcv2to1;                              // listen for communication from node 2
13       rcv {= 1: copies += dest2==1 ? data2 : 0, 1: ack1 = 2 =}
14    };
15    rcv;                                                  // slot 2: no contact
16    ... }                                  // three more slots of the same pattern
```

Listing 1. Excerpt of the Modest code for node N_1 with a bound of 2 copies

sent by N_1, and `dest1` indicates the intended receiver—both of this is to support multiple contacts per slot. As mentioned at the end of Sect. 3.2, we moved these assignments onto the `rcv` edge to make the model syntactically good for distributed scheduling; assignments prefixed `2:` are executed after prefix `1:` which follow those with no prefix. The acknowledgment mechanism uses global variables like `ack2` indicating from whom node N_2 successfully received a message. The sending node then reduces its `copies` count by that number. In slots without a contact, like slot T_2 for N_1, we just move to the next slot by a `rcv` together with the other nodes (line 15). Note that nodes cannot *create* copies; thus is because they cannot know how many copies there are at other nodes and must not violate the global bound; we can let the initial node create as many copies as allowed w.l.o.g. since any number of copies can be transmitted in a slot.

5.2 The Walker Constellation

As a realistic case study, we propose a LEO satellite constellation in a Walker formation of three orbital planes each with four equally separated satellites. The constellation is thought to provide high-latency data service to ten isolated ground nodes randomly distributed around the globe. A ground station located in Córdoba, Argentina, provides an Internet gateway that the nodes access using DTN protocols. We provide the detailed locations and parameters of all the nodes in the appendix. The ranges from ground station to satellites are set to 2000 km at a minimal elevation angle of 20° (mimicking a constrained antenna at the satellite). In turn, satellites can reach ground terminals at a distance of 1500 km at the same elevation angle. Fig. 5 shows the ground track and a 3D visualisation of the constellation. Ground nodes (user terminals) are shown in red, the ground station in green, and satellites in cyan. The domes and cones over the ground nodes and under the satellites illustrate the communication ranges. The contact plan runs for 24 hours from 01 Jul 2020 00:00:00.

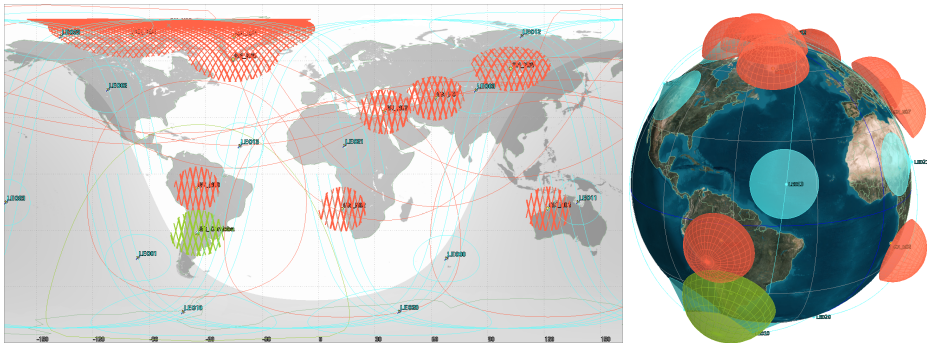


Fig. 5. Visualisation of the Walker constellation (larger version in appendix)

5.3 Experiments

We applied our toolchain to the example contact plan of Fig. 1 and the Walker constellation described above, both with at most 2 message copies in the network. For the example contact plan with unreliable communication, we can easily determine the best simple distributed scheduler and its transmission probability by again expanding the MDP semantics of Fig. 3; the probability is 0.4645. Recall from Example 4 that global-information schedulers achieve probability 0.493. We can thus compare the probabilities computed by LSS with the actual values for distributed schedulers on this example. The state space of our model of the Walker constellation is still small enough for PMC, so we can compare the values obtained via PMC and LSS for global-information schedulers at least, but not for distributed schedulers due to to the infeasibility of PMC in this case. Since the effectiveness of LSS depends on the rarity of near-optimal schedulers, we expect to see a tradeoff between LSS for global-information and LSS for distributed schedulers: There are many more global-information schedulers than distributed ones. So even though the former may realise higher probabilities, LSS might only rarely find any good global-information scheduler, whereas it may often find a good distributed scheduler due to their more limited choices.

Our experiments ran on an Intel Core i7-4790 workstation (3.6-4.0 GHz, 4 cores) with 64-bit Ubuntu Linux 18.04. We used smart sampling [**?**] with a fixed number of initial schedulers m_0 equal to the per-iteration budget n_0. In iteration i, smart sampling performs $\lceil \frac{n_i}{m_i} \rceil$ simulation runs for each of the m_i schedulers, discards the "worst" half of the schedulers according to their current probability estimate, and moves to iteration $i+1$ with $m_{i+1} = \lfloor \frac{m_i}{2} \rfloor$. We can thus cover a large number of schedulers with only $\approx \log_2(m_0) \cdot n_0$ simulation runs in total. We show the results in Table 1. We used the Modest Toolset's mcsta model checker for PMC and modes as described in Sect. 4 for the SMC-LSS-m_0 runs. Due to the randomised nature of the experiments, we repeated each three times and report the average and highest maximum probability over the repetitions. Runtimes were at most 5 minutes, for the $m_0 = 100000$ setting, with distributed scheduler runs taking around 10 % longer that those for global-

Table 1. Experimental results: average (highest) max. probabilities over 3 repetitions

	PMC	SMC-LSS-1000		SMC-LSS-10000		SMC-LSS-100000	
model	global	global	distrib.	global	distrib.	global	distrib.
example/unrel.	0.493	0.48 (0.49)	0.46 (0.47)	0.49 (0.49)	0.46 (0.47)	0.49 (0.49)	0.46 (0.46)
example/acks	0.505	0.49 (0.50)	0.46 (0.48)	0.50 (0.50)	0.50 (0.50)	0.50 (0.50)	0.50 (0.51)
walker/unrel.	0.438	0.03 (0.06)	0.21 (0.30)	0.10 (0.16)	0.30 (0.37)	0.26 (0.33)	0.37 (0.38)
walker/acks	0.734	0.36 (0.38)	0.47 (0.48)	0.38 (0.40)	0.54 (0.60)	0.45 (0.47)	0.54 (0.56)

information schedulers. We use the adaptive method of [**?**, Section III] for the statistical evaluation and request the probability for absolute error < 0.0025 on the example and < 0.005 on the Walker case to be 95 %.

We see that, on the example, both variants of LSS find respective optimal schedulers easily. With acknowledgment-based communication ("acks"), the optimal distributed scheduler curiously realises the same probability as the global one. We asked MODES to print traces under this scheduler identifier, and found that nodes N_1 and N_3 can collaborate to implement a strategy similar to the one of Example 4: In slot T_3, N_1 *always* tries to send to N_3. However, if N_3 already has a copy, it chooses `nop3` in this slot; then N_1 does not receive an acknowledgment and gets to keep its copy for sending in T_5. While "sneaky", such behaviour could clearly be implemented in satellites, and it being found validates the correctness and applicability of our distributed LSS approach. On the Walker case study, the tradeoff described earlier comes into play: distributed-scheduler LSS consistently finds better schedulers, and more reliably finds them even for lower values of m_0, despite global-information schedulers being able to realise higher probabilities in principle. In particular, our new version of LSS consistently finds schedulers not too far from the maximum achievable with global information according to PMC—but its schedulers are guaranteed implementable.

6 Conclusion

We have developed new theory and tools to tackle the challenge of computing good and implementable routing strategies for satellite DTNs under uncertain contact plans, using formal methods technology. We have proposed a new, modern notion of distributed schedulers appropriate for the application, extended the LSS technique to work with this new notion, created a toolchain incorporating the new theory and technology, and applied it to a realistic case study. While LSS may be limited (e.g. by not being able to prove safety) in a general setting, we showed that it works very well for this application—which particularly benefits from the flexibility of LSS w.r.t. handling different scheduler classes. Once we found a good scheduler, it can be implemented in a satellite by a program that feeds its identifier, the current slot offset, and the number of local copies into a reimplementation of MODES' hash function. We plan to adapt other methods (e.g. [**?**]) to extract compact human-readable descriptions of the scheduler, too.

Appendix

Walker Case Study Parameters

The tables below provide the detailed parameters of the operation segments of the mission considered in Sect. 5.2: the space segment (composed of the satellites in the constellation), the ground segment (composed of the ground station and control center) and the user segment (composed of all user terminals).

Ground Segment				
ID	Name	Latitude [deg]	Longitude [deg]	Altitude [km]
1	GS_Cordoba	-31.4135	-64.18105	0.423091597

User Segment				
ID	Name	Latitude [deg]	Longitude [deg]	Altitude [km]
2	GN_N00	-7.998	-64.739	0.101650767
3	GN_N01	40.128	62.480	0.148069703
4	GN_N02	-18.756	12.770	0.610797243
5	GN_N03	79.129	-79.583	0.558084830
6	GN_N04	72.684	-44.711	2.836686069
7	GN_N05	60.573	-45.205	0.045041046
8	GN_N06	-18.585	122.109	0.103431089
9	GN_N07	32.888	35.117	0.025320638
10	GN_N08	73.372	-98.790	0.138821220
11	GN_N09	55.670	102.459	0.466638732

Space Segment							
ID	Name	True Anomaly [deg]	Apogee Altitude [km]	Perigee Altitude [km]	Perigee Argument [deg]	Inclination [deg]	RAAN [deg]
12	LEO00	0	600	600	0	98	0
13	LEO01	90	600	600	0	98	0
14	LEO02	180	600	600	0	98	0
15	LEO03	270	600	600	0	98	0
16	LEO10	30	600	600	0	98	240
17	LEO11	120	600	600	0	98	240
18	LEO12	210	600	600	0	98	240
19	LEO13	300	600	600	0	98	240
20	LEO20	60	600	600	0	98	120
21	LEO21	150	600	600	0	98	120
22	LEO22	240	600	600	0	98	120
23	LEO23	330	600	600	0	98	120

Proof Sketch for Lemma 1

The conditions of Def. 9 only apply to states where the interleaving scheduler actually has a choice; in all others, σ_{I_1} and σ_{I_2} must coincide. So let s be a state where $\sigma_{I_1}(s) \neq \sigma_{I_2}(s)$ and let $TR_i = T(s) \cap I_t(M_{\sigma_{I_i}(s)})$ with $tr_i = s \xrightarrow{a_i} s'_i \in TR_i$ for $i \in \{1, 2\}$. Then $T(s'_1) \supseteq T(s) \setminus TR_i$ and $T(s'_2) \supseteq T(s) \setminus TR_i$, i.e. the choice of the interleaving scheduler cannot disable a non-local transition. This is because (a) condition 3 prevents the tr_i from changing the values of guards in non-involved components, and (b) condition 2 requires that, if more than one component is involved in tr_i, then $TR_i = \{ tr_i \}$. The latter ensures that taking a synchronising transition cannot disable another synchronising transition by taking away a needed "synchronisation partner" (a transition with the same label as needed by the synchronisation vectors). Thus the restriction of $[\![M]\!]$ induced by an interleaving scheduler together with condition 1 and the fairness requirement is a (stronger) variant of a partial order reduction using ample sets as in [**?**] (with condition **A5'**).

Larger Version of the Graphics of Sect. 5.2