Modeling Hybrid Systems with Stochastic Events in HYPE

Luca Bortolussi
Department of Maths and Computer Science, University of Trieste.
luca@dmi.units.it

Vashti Galpin   Jane Hillston
Laboratory for Foundations of Computer Science, University of Edinburgh
Vashti.Galpin@ed.ac.uk   Jane.Hillston@ed.ac.uk

The process algebra HYPE was recently proposed as a fine-grained modelling approach for capturing the behaviour of hybrid systems. In the original proposal, the semantics of HYPE was defined in terms of (non)deterministic hybrid systems. Here we extend the language adding stochastic events, hence obtaining a semantics in terms of Transition Driven Stochastic Hybrid Automata, a subset of a general class of stochastic process termed Piecewise Deterministic Markov Processes. The definition of stochastic HYPE is discussed by means of an example of a delay tolerant network.

Many natural and engineered systems present a hybrid dynamical behaviour, where periods of continuous evolution are punctuated by the happening of discrete events, which can alter the course of subsequent continuous evolution. As an example, consider a thermostatically controlled heater, when working. The air temperature is a continuous quantity, subject to thermodynamic cooling and to the heating effect of the heater. The discrete events that alter the temperature dynamics are the turning on or off the heater by the thermostat in response to the air temperature. A different example is that of genetic regulatory networks: Genes, considered as single entities, can be seen as switches, which are active only when the concentration of transcription factors regulating their expression is above or below a gene-specific threshold. The behaviour of such systems can be captured by a collection of sets of ODEs, with discrete events shifting the dynamic behaviour from the control of one set of ODEs to another. This is the approach taken with hybrid automata [6].

Recently, there has been a growing interest in the definition of process algebras for hybrid settings. The many hybrid process algebras proposed [7] differ substantially in syntax, semantics, discontinuous behaviour, flow-determinism, theoretical results and availability of tools. However, they are all similar in their approach in that the dynamic behaviour of each component must be fully described in terms of a set of ODEs given explicitly in the syntax of the process algebra.

HYPE, a hybrid process algebra we recently proposed [4, 5], is distinguished in that it captures behaviour at a fine-grained level, composing distinct flows or influences which act on the continuous variables of the system. At a superficial level this removes the need to explicitly write ODEs in the process algebra syntax. Instead the dynamic behaviour emerges, via the semantics of the language, when these elements are composed. Moreover the use of flows as the basic elements of model construction has advantages such as ease and simplification of modelling. This approach assists the modeller in allowing them to identify smaller or local descriptions of the model and then to combine these descriptions to obtain the larger system. The explicit controller also helps to separate modelling concerns.

In the original definition of HYPE, discrete events are *urgent*, meaning that they happen instantaneously as soon as their activation condition, a condition on the values of continuous variables which are evolving in the system, becomes true. However, to deal with events which are not so tightly tied to the continuous evolution of the system and may appear to occur randomly, we introduced *non-urgent* events,

Submitted to:
PASTA 2010

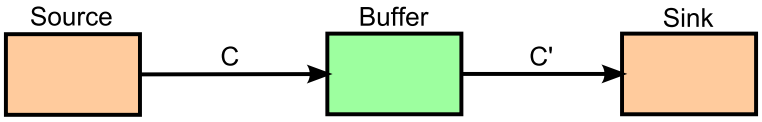

Figure 1: Schematic model of a very simple DTN, with just one node.

having an unspecified activation condition $\perp$. Such events happen non-deterministically in time, without further constraints. In order to carry out more precise quantified analysis of the constructed model, we refine this notion of non-urgent events, by introducing *stochastic actions*. These actions will have an activation condition which is a random variable, capturing the probability distribution of the time of occurrence of the event.

This small modification substantially enriches the class of underlying mathematical processes which capture the systems in HYPE. The previous version of HYPE had a semantics in terms of hybrid automata [6]. Now its semantics is given in terms of Piecewise Deterministic Markov Processes (PDMPs) [2]. This semantics is constructed in two steps, by first mapping a HYPE model to an automata based formalism, namely Transition Driven Stochastic Hybrid Automata [1], which are subsequently mapped to PDMPs.

In the following, we will present the stochastic version of HYPE by means of a running example. In particular, we will show the syntax and the dynamics of a simple model of a Delay Tolerant Network (DTN). The definition of the semantics and other formal issues will be presented in a future paper. More details about the non-stochastic version of the language can be found in [4, 5].

DTN model in HYPE

Delay Tolerant Networks are communication networks whose nodes have an intermittent connectivity. As packets cannot always be forwarded, nodes need additional space for storage [3]. The simple model we consider is shown in Figure 1. There is one single intermediate buffer node between the source and the sink, having a limited buffer size. Packets are dropped when the buffer is full.

HYPE modelling style is based on few basic principles: First, the description of the different flows and the way they react to events (*uncontrolled system*) is separated from the description of the control structure of discrete events (*controller*). Moreover, the uncontrolled system is the composition of components and subcomponents, with each component being one entity of the model and each subcomponent being the description of a single flow source acting on a component. For instance, the buffer is a component of the HYPE model of the DTN, and it is composed by two subcomponents, one describing the reception of packets (input subcomponent) and the other describing their dispatch (output subcomponent).

The input subcomponent is the following:

$$\begin{aligned} I_{C,B} \overset{def}{=} \;& \overline{\mathrm{on}}_C{:}(i_B, s_C, c).I_{C,B} + \overline{\mathrm{off}}_C{:}(i_B, 0, c).I_{C,B} + \\ & \underline{\mathrm{full}}_B{:}(i_B, 0, c).I_{C,B} + \underline{\mathrm{init}}_B{:}(i_B, s_C, c).I_{C,B} + \\ & \underline{\mathrm{nf\text{-}on}}_B{:}(i_B, s_C, c).I_{C,B} + \underline{\mathrm{nf\text{-}off}}{:}(i_B, 0, c).I_{C,B} \end{aligned}$$

It is the summation of different addends, each representing the reaction of the subcomponent to a specific event. For instance $\underline{\mathrm{full}}_B{:}(i_B, 0, c).I_{C,B}$ states that, when the buffer is full, the input flow will be governed by the tuple $(i_B, 0, c)$, called an *activity*. Activities describe flows, and they are composed of three parts:

an *influence name*, i_B, essentially identifying the continuous variable affected by the flow (B in this example), an *influence strength*, 0 in this case, and in *influence type* (c in the example), describing the functional dependence of the flow on other variables (in this case, the flow is constant). The specific activity $(i_B, 0, c)$ basically states that the input flow will be stopped when the buffer is full.

Event $\underline{\mathrm{full}}_B$ is *urgent*, meaning that it happens deterministically. Its *event condition* consists of two parts: a *guard* or *activation condition* and a *reset*. The guard is a formula on continuous variables: when it evaluates to true, the event happens. Resets, instead, allow the instantaneous modification of the values of variables. In particular, the activation condition for $\underline{\mathrm{full}}_B$ is $B = max_B$, meaning that the $\underline{\mathrm{full}}_B$ event is fired as soon as the buffer reaches its maximum capacity. Furthermore, its reset is *true*, meaning that no variable will be modified. Other urgent events are $\underline{\mathrm{nf\text{-}on}}_B$ and $\underline{\mathrm{nf\text{-}off}}_B$, which are triggered when the buffer stops being full, depending on the status of the incoming link (i.e. $\underline{\mathrm{nf\text{-}on}}_B$ fires if $B < max_B$ and the incoming link is working).

The events $\overline{\mathrm{on}}_C$ and $\overline{\mathrm{off}}_C$ model the activation or deactivation of the link, and they are stochastic. Essentially, their activation condition is not a formula, but rather the rate of the exponentially distributed random variable modelling the time in which the events happen. Rates can be functions of the continuous variables, although in this case the rate is constant: $r_{on} \in \mathbb{R}^+$ for $\overline{\mathrm{on}}_C$ and $r_{off} \in \mathbb{R}^+$ for $\overline{\mathrm{off}}_C$. The event $\underline{\mathrm{init}}_B$ is a special event used to set up the initial state of the system.

The output subcomponent is defined in a similar way, with C' representing the outgoing link:

$$\begin{aligned} O_{B,C'} \overset{def}{=} & \overline{\mathrm{on}}_{C'}{:}(o_B, -t_{C'}, c).O_{B,C'} + \overline{\mathrm{off}}_{C'}{:}(o_B, 0, c).O_{B,C'} + \\ & \underline{\mathrm{empty}}_B{:}(o_B, 0, c).O_{B,C'} + \underline{\mathrm{init}}_B{:}(o_B, -t_{C'}, c).O_{B,C'} + \\ & \underline{\mathrm{ne\text{-}on}}_B{:}(o_B, -t_{C'}, c).O_{B,C'} + \underline{\mathrm{ne\text{-}off}}{:}(o_B, 0, c).O_{B,C'} \end{aligned}$$

The buffer component is then obtained by composing the input and the output subcomponents, synchronizing them on their shared events (just $\underline{\mathrm{init}}$, in this case):

$$Buff_{C,B,C'} \overset{def}{=} I_{C,B} \underset{\underline{\mathrm{init}}}{\bowtie} O_{B,C'}$$

In each instant, only one activity with name i_B and one with name o_B will be active in the system, determining the continuous evolution of variables. For example, if the buffer is not full and its incoming and outgoing links are working, the enabled activities will be (i_B, s_C, c) and $(o_B, -t_C, c)$. They give rise to an ODE for the buffer capacity B, which is obtained by adding the flows described by the activities. In this case, $\dot{B} = s_C - t_C$.

The other part of the DTN model is the *controller*, which imposes causality on the happening of discrete events. Controllers are usually the composition of different sub-controllers, each regulating a subset of events. This is the case also for a single buffer in the DTN model, whose controller is

$$Con_{C,B,C'} \overset{def}{=} ConI^1_{C,B} \underset{\emptyset}{\bowtie} ConO^1_{B,C'}$$

where $ConI^1_{C,B}$ controls the input and $ConO^1_{B,C'}$ controls the output. More specifically,

$$\begin{aligned} ConI^0_{C,B} &\overset{def}{=} \overline{\mathrm{on}}_C.ConI^1_{C,B} \\ ConI^1_{C,B} &\overset{def}{=} \overline{\mathrm{off}}_C.ConI^1_{C,B} + \underline{\mathrm{full}}_B.ConI^2_{C,B} \\ ConI^2_{C,B} &\overset{def}{=} \overline{\mathrm{off}}_C.ConI^3_{C,B} + \underline{\mathrm{nf\text{-}on}}_B.ConI^1_{C,B} \\ ConI^3_{C,B} &\overset{def}{=} \overline{\mathrm{on}}_C.\underline{\mathrm{full}}_B.ConI^2_{C,B} + \underline{\mathrm{nf\text{-}off}}_B.ConI^0_{C,B} \end{aligned}$$

This controller states that when the incoming link is working ($ConI^1_{C,B}$), it may stop working or the buffer may become full. When it is full, it may stop working or stop being full, and so on. Notice that it is the

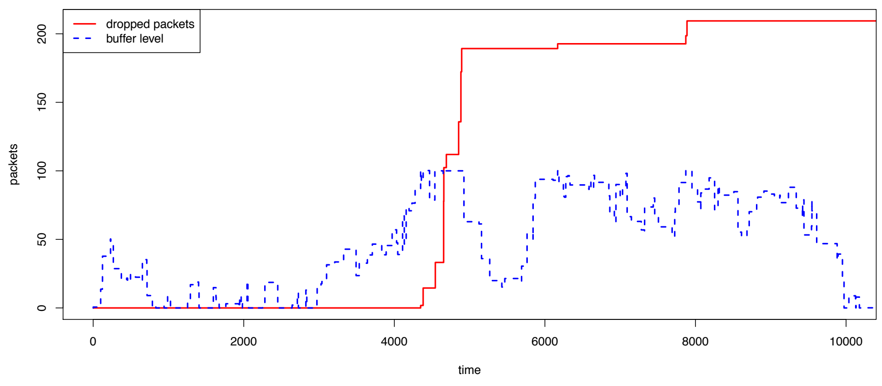

Figure 2: Simulated trajectory of buffer size of the the DTN. The on and off rates of input and output links are the same, and off rates is four order of magnitude bigger than the on rate. Max buffer size is 100.

controller that “remembers” if the link was working or not when the buffer changes from full to non-full. The output controller is defined in a similar way.

In Figure 2, we show a simulated trajectory of the buffer level of the DTN and of the number of dropped packets, assuming a maximum buffer size of 100. Figure 3, instead, shows a simulated trajectory for a buffer with maximum size equal to 150. In the latter case, the number of dropped packets is considerably less, as can be expected.

Conclusions

We presented a stochastic version of HYPE, in which non-urgent events are fired according to an exponentially distributed random time. The hybrid models so obtained are Piecewise Deterministic Markov Processes, whose dynamics is an alternation of periods of continuous evolution and discrete jumps, either forced or stochastic.

We introduced stochastic HYPE by means of an example, discussing the model of a simple delay tolerant network. We also showed two simulated trajectories of the system, to provide a flavor of the dynamics of the model.

Given a stochastic hybrid model, we can perform different kind of analyses, from simulation to reachability computations and model checking, using both numerical or statistical methods.

The DTN model itself deserves more attention, and will be studied in further detail in the future, assuming more interesting network topologies with more internal nodes.

References

[1] L. Bortolussi and A. Policriti. Hybrid Semantics of Stochastic Programs with Dynamic Reconfiguration. In *Proceedings of CompMod 2009*, 2009.

[2] M.H.A. Davis. *Markov Models and Optimization*. Chapman & Hall, 1993.

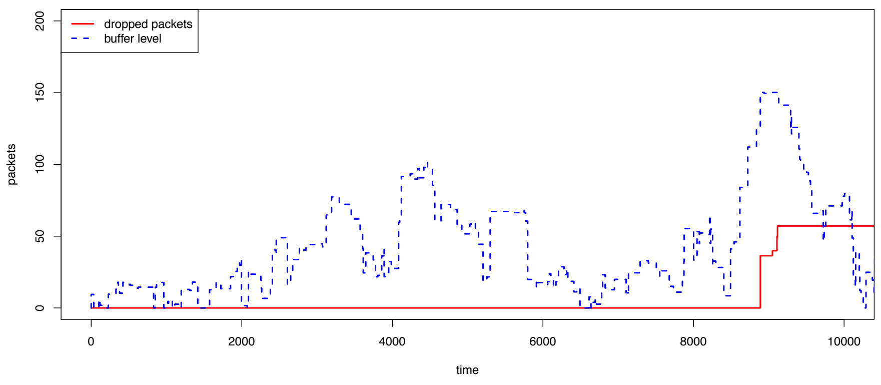

Figure 3: Simulated trajectory of buffer size of the the DTN. Max buffer size is 150.

[3] K. Fall and S. Farrell. DTN: An architectural retrospective. *IEEE Journal on Selected Areas in Communications*, 26, 2008.

[4] V. Galpin, J. Hillston, and L. Bortolussi. HYPE: a process algebra for compositional flows and emergent behaviour. In: *Proceedings of CONCUR 2009, Lecture Notes in Computer Science*, 5710:305–320, 2009.

[5] V. Galpin, J. Hillston, and L. Bortolussi. HYPE: hybrid modelling by composition of flows. *Journal version, in preparation*.

[6] T. A. Henzinger. The theory of hybrid automata. In *LICS*, pages 278–292, 1996.

[7] U. Khadim. A comparative study of process algebras for hybrid systems. Computer Science Report CSR 06-23, Technische Universiteit Eindhoven, 2006. `http://alexandria.tue.nl/extra1/wskrap/publichtml/200623.pdf`.