Stochastic Analysis of Graph Transformation Systems: A Case Study in P2P Networks

Reiko Heckel*

Department of Computer Science
University of Leicester
United Kingdom

Abstract. In distributed and mobile systems with volatile bandwidth and fragile connectivity, non-functional aspects like performance and reliability become more and more important. To formalise, measure, and predict these properties, *stochastic methods* are required. At the same time such systems are characterised by a high degree of architectural reconfiguration. Viewing the architecture of a distributed system as a graph, this is naturally modelled by *graph transformations*.
To address these two concerns, stochastic graph transformation systems have been introduced associating with each rule its application rate—the rate of the exponential distribution governing the delay of its application. Deriving continuous-time Markov chains, Continuous Stochastic Logic is used to specify reliability properties and verify them through model checking.
In particular, we study a protocol for the reconfiguration of P2P networks intended to improve their reliability by adding redundant connections. The modelling of this protocol as a (stochastic) graph transformation system takes advantage of negative application and conditions path expressions. This ensuing high-level style of specification helps to reduce the number of states and increases the capabilities for automated analysis.

1 Introduction

Non-functional requirements, concerning the quality or resources of a system, are often difficult to capture, measure, and predict. At the same time they are usually critical for success. Many failures of software engineering projects have been attributed to a lack of understanding of non-functional aspects in the early stages of development [9].

With the success of Internet and mobile technology, properties like the reliability of connections, available bandwidth and computing resources become an even greater concern. Since individual occurrences of failures are generally unpredictable, stochastic concepts are required to formalise such properties. Many specification formalisms provide corresponding extensions, including stochastic transition systems (or Markov chains [2, 21]), stochastic Petri nets [1, 4, 19, 20] or

* European Community's Human Potential Programme under contract HPRN-CT-2002-00275, [SegraVis]

process algebras [5, 7]. Most of these formalisms specialise in describing behaviour in terms of orderings of events, neglecting aspects like data transformations and changes to software architecture or network topology.

A noticeable exception is the π-calculus [18], which allows communication of channel names between interacting processes. It is thus possible to describe changes of data structures or network topologies. The stochastic π-calculus [22], extending the original by assigning rates to the communication actions of a process, allows to address non-functional aspects. However, while the π-calculus is an adequate semantic framework for programming, it is too low-level for expressing requirements in the early stages of a project. Here communication between developers and clients requires a direct, diagrammatic description of *what* changes are required, instead of a detailed description of *how* they are achieved.

A more abstract style of specification is provided by rewriting-based formalisms like Rewriting Logic or Graph Transformation [17, 24]. Here, rules specify pre- and post-conditions of operations (*what* should be achieved) in terms of complex patterns, while the underlying mechanisms for pattern matching and implementing these changes are hidden from the user. Graph transformation, in particular, supports a visual representation of rules which is reminiscent of to the intuitive way in which engineers would sketch, for example, network reconfigurations.

In order to account for the non-functional aspects, we introduced *stochastic* graph transformation systems [11]. Associating an exponentially distributed application delay with each rule, we derive continuous-time Markov chains (CTMCs), the standard model for stochastic analysis. This enables us to establish a link to continuous stochastic logic (CSL) to express and verify properties like the probability of being connected within 20 seconds after start-up, the long-term probability for connectedness, etc.

This paper is devoted to a case study, a simplified version of a protocol for the reconfiguration of Peer-to-Peer networks [16], to validate the practicability of the approach. P2P networks are decentralised overlay networks that use a given transport infrastructure like the Internet to create a self-organising network. Due to the lack of global control and potential unreliability of the infrastructure, P2P networks are prone to dependability problems. The standard solution consists in creating sufficient redundancy so that, when a node unexpectedly leaves the network, its role in the routing of information can be taken over by other nodes.

Mariani [16] proposes an algorithm which, executed asynchronously by each peer, adds redundant connections to the network to guarantee that the disappearance of a peer does not unduly affect the overall performance and routing capabilities of the network. It does so by querying the local context of a node up to a given depth to expose potential weaknesses in the network topology. The assumption is that this happens fast enough to prevent loss of connectivity due to the disappearance of the node before extra links could be added. The desired result is an increased fault tolerance.

We are going to validate these assumptions and compare the level of fault tolerance achieved with the one obtained by the simpler solution of just adding

a limited number of references at random. To this purpose, we shall model the protocol as a stochastic graph transformation systems and analyse different variants of it. To develop a satisfactory model, we will require advanced features for controlling the application of rules, like negative application conditions and path expressions. We give an introduction to the basic approach and its extensions and discuss their relevance w.r.t. the model checking problem.

The paper is structured as follows. Below, in Sect. 2 we introduce typed graph transformation systems and provide a functional model model of the P2P network. In Sect. 3 we extend definitions and examples to stochastic graph transformation systems, including the derivation of Markov chains, stochastic logic and model checking. Their application to the case study is reported in Sect. 4. Sect. 5 concludes the paper with a discussion of tools and relevant theoretical problems.

2 Graph Transformation Systems

In this section we will first focus on the basic ideas of typed graph transformation systems, followed by a survey of the more advanced concepts required by our case study. We follow the so-called algebraic single-pushout (SPO) approach [15] to the transformation of typed graphs [13, 6].

2.1 Type and Instance Graphs

Graphs provide the most basic mathematical model for entities and relations. A graph consists of a set of vertices V and a set of edges E such that each edge e in E has a source and a target vertex $s(e)$ and $t(e)$ in V, respectively.

In this paper, graphs shall represent configurations of a Peer-to-Peer (P2P) network, modelling network nodes as vertices and links between them as edges. We distinguish two different kinds of nodes in our networks, labelled by P for peers and R for registry, as well as edge types l and r representing links and registrations, respectively. The idea is that new peers participating in the network have to login with a central registry server. Afterwards, they can connect and communicate directly, without using any central infrastructure.

The graph in the upper right of Fig. 1 represents a network with a single participant and the registry, while the one in the upper left has two connected participants. Our graphs are directed, but in the case of links we use undirected l-edges edges to denote symmetric pairs of directed ones.

Like a network configuration, also a collection of interrelated types may be represented as a graph. In the bottom, Figure 1 shows the type graph of the P2P model, providing the types for the instance graphs in the top. The relation between types and their occurrences in configurations is formally captured by the notion of *typed graphs*: A fixed *type graph* TG represents the type level and its *instance graphs* the individual snapshots.

Definition 1 (typed graphs). *A* directed (unlabelled) graph *is a four-tuple* $G = \langle G_V, G_E, src^G, tar^G \rangle$ *with a set of vertices* G_V*, a set of edges* G_E*, and*

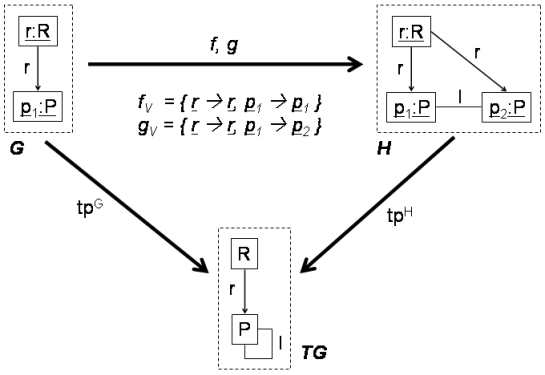

Fig. 1. Type and instance graphs

functions $src^G : G_E \to G_V$ *and* $tar^G : G_E \to G_V$ *associating to each edge its source and target vertex. A graph homomorphism* $f : G \to H$ *is a pair of functions* $\langle f_V : G_V \to H_V, f_E : G_E \to H_E \rangle$ *preserving source and target, i.e., such that* $f_V \circ src^G = src^H \circ f_E$ *and* $f_V \circ tar^G = tar^H \circ f_E$.

Fixing a type graph TG*, an* instance graph $\langle G, tp^G \rangle$ *over* TG *is a graph* G *equipped with a graph homomorphism* $tp^G : G \to TG$*. A* morphism of typed graphs $h : \langle G_1, g_1 \rangle \to \langle G_2, g_2 \rangle$ *is a graph homomorphism* $h : G_1 \to G_2$ *that preserves the typing, that is,* $tp^{G_2} \circ h = tp^{G_1}$.

We us the notation of the Unified Modelling Language (UML) for class and object diagrams to capture the distinction between types and instances: $\underline{r : R}$ denotes an element of an instance graph $\langle G, tp^G \rangle$ such that its type $tp^G(r) = R$. The expression is underlined to stress that it is considered part of a system configuration (rather than a rule as we shall see below). Morphisms between typed graphs $\langle G, tp^G \rangle$ and $\langle H, tp^H \rangle$ are exemplified in Fig. 1. Morphism f represents a subgraph inclusion while g, combining inclusion and renaming, is an injective homomorphism or subgraph isomorphism.

2.2 Single-Pushout Graph Transformation

Having modelled configurations as instance graphs, we are turning to the specification of instance graph transformations by means of rules. A rule can be seen as a representative example of all transformations, modelling them by means of patterns for pre and post states.

For a given type graph TG, a *graph transformation rule* $p : L \to R$ consists of a name p and a pair of graphs typed over TG. The left-hand side L represents the pre-condition of the operation specified by the rule while the right-hand side R describes the post-condition. A correspondence between elements in L and R

is given by the identities of the nodes (sometimes omitted, assuming that the intention is obvious from the layout).

The rules for the P2P network model are shown in Fig. 2 and 5. Rule *new* creates a new peer. This requires to look up the registration of an existing peer at the registry server, represented by the r-edge from $r : R$ to $p : P$, to create a new peer $p_1 : P$ with corresponding registration, and to link it to p with a new edge of type l.

Rule *kill* models the deletion of a peer with all its ingoing and outgoing edges. This may cause the network to become disconnected, except for registrations, which are not used for communication. The rule *disconnected* in the bottom is provided to detect such situations. The rule is applicable if there are two registered nodes which are *not connected* by a path of l-edges, but the application does not have any effect on the graph. This rule combines two interesting features: Negative application conditions and path expressions, both to be introduced below in more detail.

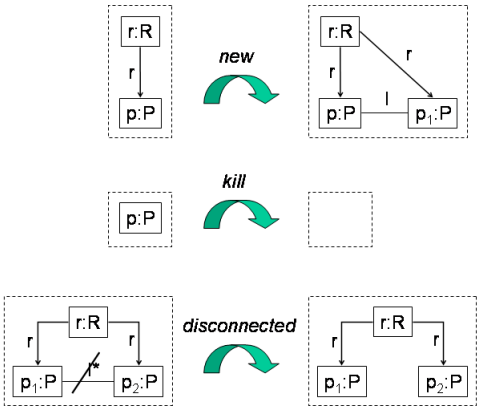

Fig. 2. Rules for creating and killing peers

Rules generate transformations by replacing in a given graph a match for the left-hand side with a copy of the right-hand side. Thus, a *graph transformation* from a pre-state G to a post-state H, denoted by $G \overset{p(m)}{\Longrightarrow} H$, is performed in three steps.

1. *Find* a match of the left-hand side L in the given graph G, represented by an injective graph morphism $m : L \to G$, and check if it satisfies the application conditions, if any;
2. *Delete* from G all vertices and edges matching $L \setminus R$;
3. *Paste* to the result a *copy* of $R \setminus L$, yielding the derived graph H.

In Fig. 3 the application of a rule is shown which creates a new peer, but unlike *new* in Fig. 2 passes on the registration from the existing to the new peer.

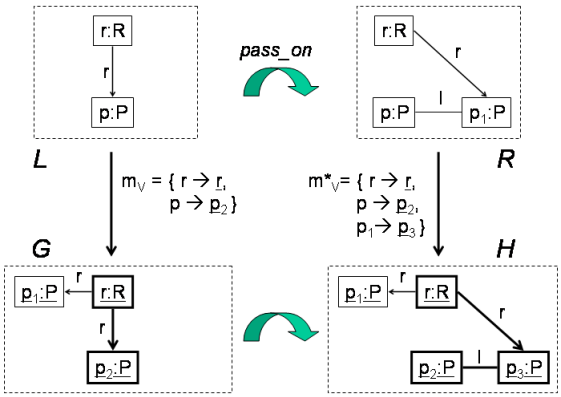

Fig. 3. Transformation step using rule *collect*

The match m of the rule's left-hand side is indicated by the boldface nodes and edges in G. The transformation deletes the r-edge from $\underline{r : R}$ to $\underline{p_2 : P}$, because it is matched by an edge in the left-hand side L, which does not occur in R. To the graph obtained after deletion, we paste a copy of the node $p_1 : P$ in L, renaming it to $\underline{p_3}$ to avoid a name conflict, as well as copies of the l-edge from $p : P$ to $p_1 : P$ and the r-edge from $r : R$ to p_1. The match m tells us where these edges must be added, e.g., $p \mapsto \underline{p_2}$ means that the new l-edge is attached to $\underline{p_2}$ rather than to $\underline{p_1}$ in H. However, this is not the only possibility for applying this rule. Another option would be to match p by $\underline{p_1}$, attaching the link to a different peer. That means, there are two causes of non-determinism: choosing the rule to be applied (e.g., *new* or *pass_on*) and the match at which it is applied. (In this case, both transformations lead to graphs that are isomorphic, i.e., differ only up to renaming.)

The example of Fig. 3 is not entirely representative of the problems that may be caused by deleting elements in a graph during step 2. In fact, we have to make sure that the remaining structure is still a valid graph, i.e., that no edges are left dangling because of the deletion of their source or target vertices. The problem is exemplified by the step in Fig. 4. The deletion of $\underline{p_2 : P}$ would leave the attached r and l edges "dangling".

There exist two solutions to this problem: a radical and a conservative one. The first gives priority to deletion, removing the vertex along with the dangling edges. The conservative alternative consists in assuming a standard applications condition which excludes the depicted situation as valid transformation. This application condition is known as the *dangling condition*, and it is characteristic of the algebraic DPO (double-pushout) approach to graph transformation [8].

We adopt the more radical Single-Pushout (SPO) approach [15] because it provides a more realistic representation of the behaviour we intend to model: It may not be possible to stop a peer from leaving the network, even if it is still

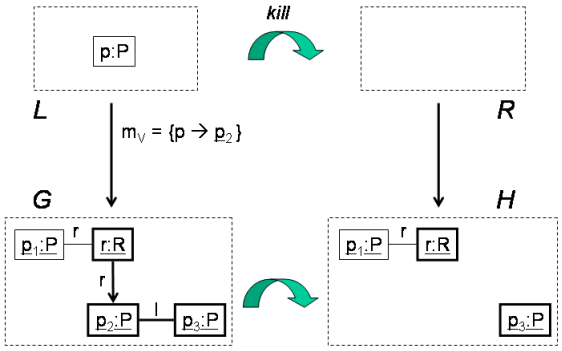

Fig. 4. More interesting example

connected to other peers. The SPO approach owes its name to the fact that the construction of applying a transformation rule can be formalised as a pushout (a gluing construction) in the category of graphs and partial graph homomorphisms [15]. A partial graph morphism $g : G \to H$ is a total morphisms from some subgraph $dom(g)$ of G to H. We consider the simplified case of injective matching, where the left-hand side is essentially a subgraph of the graph to be transformed, rather than an arbitrary homomorphic image.

Definition 2 (rule, match, transformation). *A* rule $p : L \xrightarrow{r} R$ *consists of a rule name* p *and a partial graph morphism* r*. A* match *for* $r : L \to R$ *into some graph* G *is a total injective morphism* $m : L \to G$*. Given a rule* p *and a match* m *for* p *in a graph* G *the* direct (SPO-) transformation *from* G *with* p *at* m*, written* $G \overset{p(m)}{\Longrightarrow} H$*, is the pushout of* r *and* m *in the category of graphs and partial graph morphisms.*

$$\begin{array}{ccc} L & \xrightarrow{\quad r \quad} & R \\ {\scriptstyle m}\downarrow & (1) & \downarrow{\scriptstyle m^*} \\ G & \xrightarrow[\quad r^* \quad]{} & H \end{array}$$

The typing $G, L \xrightarrow{r} R$, and $L \xrightarrow{m} G$ over TG induces a unique typing for the derived graph H as well as for the tracking morphism r^* and the co-match m^*. Intuitively, all elements that are preserved get their typing from G via r^* and all new elements inherit their typing from R via m^*. Pushout properties of (1) imply that there are no further elements in H (i.e., r^* and m^* are jointly surjective) and for all elements that are in the image of both morphisms, there exists a common pre-image in L so that commutativity of the diagram and type compatibility of r and m ensure that they inherit the same types from R and G.

2.3 Application Conditions and Path Expressions

Quite often, plain graph matching is not enough to express sophisticated application conditions. An example is the dangling edge condition, requiring that there are no edges incident to nodes that are to be deleted, except for those that are already part of the rule.

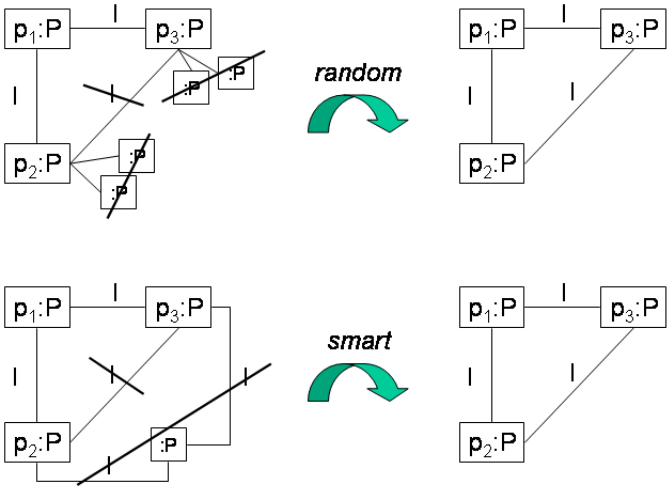

Fig. 5. Rules for introducing short-cuts in the network

User defined negative application conditions [10] can "sense" the existence or non-existence of connections in the vicinity of the match. As examples, Fig. 5 shows the rules for creating redundant links in the network to achieve a higher fault tolerance in case a node is unexpectedly deleted. Using *smart* in the bottom, a shortcut is introduced if the two neighbours of a peer are not otherwise connected by a direct link or via a third peer. This is expressed by two negative context conditions: the crossed out l-edge and the crossed out P-node with its two attached edges.

The rule should be applicable at match m only if m can *not* be extended to include any of the two forbidden structures, i.e., neither the crossed out l-edge nor the P-node with its two edges. They are represented in Fig. 6 by two injective morphisms l_1 and l_2 outgoing from the left-hand side L. Extension l_i is present in graph G if an injective morphism n_i can be found which coincides with m on L, i.e., the corresponding sub-diagram commutes.

Definition 3 (application conditions). *A* constraint over L *is an injective typed graph morphism* $L \xrightarrow{l} \hat{L}$. *Given a a match (injective morphism)* $L \xrightarrow{m} G$,

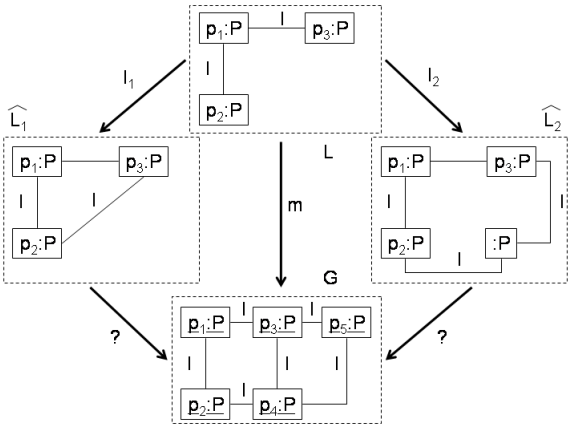

Fig. 6. Satisfaction of shortcut constraints as graph morphisms

match m satisfies l*, written* $m \models_L l$*, if there is an injective morphism* $\hat{L} \xrightarrow{n} G$*, such that* $n \circ l = m$*. An* application condition *is a Boolean expression using constraints over* L *as atomic propositions. Satisfaction is defined as usual, based on the satisfaction of constraints.*

A conditional transformation step is a transformation step where the match satisfies the application conditions associated with the rule.

The negative application in Fig. 6 is thus of the structure $N = \neg l_1 \wedge \neg l_2$. Its satisfaction does not only depend on the graph G, but also on the chosen match m. Consider, for example, $m_1 = \{p_i \mapsto \underline{p_i}\}$, $m_2 = \{p_1 \mapsto \underline{p_3}, p_2 \mapsto \underline{p_4}, p_3 \mapsto \underline{p_5}\}$, and $m_3 = \{p_1 \mapsto \underline{p_3}, p_2 \mapsto \underline{p_1}, p_3 \mapsto \underline{p_5}\}$. Then $m_1, m_2 \not\models_L N$, but $m_3 \models_L N$.

The rule *random* in Fig. 5 models the naive approach of adding links at random as long as the number of additional l-edges attached to either p_3 or p_4, beyond the ones linking them to p_1, do not exceed two. Hence, the rule will not increase the degree of any node beyond three. This condition is expressed by negative constraints, too. Note that injectivity of $\hat{L} \xrightarrow{n} G$ is essential here, because this enables us to count the number of nodes in a graph which would have been confused otherwise.

Path expressions specifying the (non-)existence of certain paths support the navigation within graphs and are generally useful if non-local graph properties shall be expressed. For instance, rule *disconnected* in Fig. 2 detects disconnected parts of the graph.

For vertices $v, w \in G_V$, a *path* from v to w is a sequence of edges $s = (e_1, e_2, \ldots, e_n) \in G_E$ such that $tar^G(e_i) = src^G(e_{i+1})$ for all $i \in \{1, \ldots, n-1\}$ (the target vertex of one edge is the source of its successor), $v = src^G(e_1)$ and $w = tar^G(e_n)$. If G is typed over TG by tp^G, the type of s is is defined by extending tp^G to sequences, i.e., $tp^G(s) = tp^G_E(e_1), tp^G_E(e_2), \ldots, tp^G_E(e_n)$.

A path expressions p is a regular expression over the alphabet TG_E of edge types. Labelling an edge e in the left-hand side of a rule, it is satisfied by a match

$m : L \to G$ if there exists a path s from $m(src^L(e))$ to $m(tar^L(e))$ such that $tp^G(s) = p$.

Path expressions are formally subsumed by Def. 3 if we allow for a countably infinite set of constraints and infinitary Boolean expressions as application conditions. An expression stating the non-existence of a path labelled by l-edges, like in rule *disconnected* in Fig. 2, is then represented by a conjunction $\neg l_1 \wedge \neg l_2 \wedge \ldots$ where the l_i are constraints specifying paths of length i.

2.4 Graph Transformation Systems

Rules over the same type graph are collected in a graph transformation system. Given a graph to start with, they can generate any of the usual state-based models, like sets of traces, labelled transition systems, event structures. We will be particularly interested in a variant of transition systems.

Definition 4 (graph transformation system). *A* graph transformation system $\mathcal{G} = \langle TG, P \rangle$ *consists of a type graph* TG *and a set of (conditional) graph transformation rules* $p : L \xrightarrow{r} R \in P$. *The application condition of* p *is denoted by* $AP(p)$.

A transformation sequence in $\mathcal{G}$

$$G_0 \overset{p_1(m_1)}{\Longrightarrow} G_1 \overset{p_2(m_2)}{\Longrightarrow} \cdots \overset{p_k(m_k)}{\Longrightarrow} G_k$$

is a sequence of consecutive transformation steps with $p_i \in P$, *briefly denoted by* $G_0 \Longrightarrow^*_{\mathcal{G}} G_k$.

The graph transformation systems we shall be interested in are

- $\mathcal{G}_{random} = \langle TG, \{new, kill, disconnected, random\} \rangle$
- $\mathcal{G}_{smart} = \langle TG, \{new, kill, disconnected, smart\} \rangle$

with TG being the type graph shown in the bottom of Fig. 1 and the rules given in Fig. 2 and 5.

A labelled transition graph is the multi-graph equivalent of a labelled transition system, allowing for more than one transition between a given pair of states, defined as isomorphism classes of the graphs reachable from the initial one.

Definition 5 (induced labelled transition graph). *Let* $\mathcal{G} = \langle TG, P \rangle$ *be a graph transformation system and* G_0 *a graph typed over* TG. *Assume a fixed mapping* χ *associating to each isomorphism class* C *of typed graphs a representative* G, *i.e.* $\chi(C) = G$ *with* $C = [G] := \{H \mid H \cong G\}$. *The* labelled transition graph induced by $\mathcal{G}$ and G_0 *is given by* $LTG(\mathcal{G}, G_0) = \langle L, S, T, pre, post, lab \rangle$, *where*

- $L = P$ *is the set of rule names of* $\mathcal{G}$;
- S *is the set of all isomorphism classes of graphs reachable from* G_0, *i.e.* $S = \{\, [G_n] \mid G_0 \Longrightarrow^*_{\mathcal{G}} G_n \}$;

– *T is the set of transformations $t = (G \overset{p(m)}{\Longrightarrow} H)$ with $\chi(s) = G$ and $\chi(s') = H$ for some $s, s' \in S$. In this case, $pre(t) = s$, $post(t) = s'$, $lab(t) = p$ and we write briefly $s \overset{p}{\Longrightarrow} s'$.*

Multiple transitions are of interest when in the following section labelled transition graphs are used to derive Markov chains.

3 Stochastic Graph Transformation

In this section, we introduce stochastic graph transformations extending typed graph transformation systems in the SPO approach by rates associated with rule names. We show how to derive a Continuous-Time Markov Chain (CTMC) from the generated transition system, thus providing the basis for stochastic logic and model checking in Section 3.3.

3.1 Markov Chains

First we provide some basic notions adopting the Q-matrix, a kind of "incidence matrix" of the Markov Chain, as elementary notion (cf. [21]).

Definition 6 (Q-matrix). *Let S be a countable set. A Q-matrix on S is a real-valued matrix $Q = Q(s, s')_{s,s' \in S}$ satisfying the following conditions:*

(i) $0 \leq -Q(s, s) < \infty$ *for all* $s \in S$,
(ii) $Q(s, s') \geq 0$ *for all* $s \neq s'$,
(iii) $\sum_{s' \in S} Q(s, s') = 0$ *for all* $s \in S$.

The Q-matrix is also called *transition rate matrix*. We use Q-matrices in order to define random processes. A random process is a family of random variables $X(t)$ where t is an indexing parameter. Depending on whether t is taken from a discrete or continuous set, we speak of a discrete- or continuous-time process, respectively.

We consider continuous-time random processes in which the number of times the random variables $X(t)$ changes value is finite or countable. Let $t_1, t_2, t_3, \ldots$ be the times at which the state changes occur. If we ignore how long the random process remains in a given state, we can view the sequence $X(t_1), X(t_2), X(t_3), \ldots$ as a discrete-time process embedded in the continuous-time process, the so called *jump chain* [21, 2.2].

Definition 7 (CTMC). *A continuous-time Markov chain (CTMC) is a continuous-time, discrete-state random process such that*

1. *The jump chain is a discrete-time Markov chain, i.e. a random process in which the current state depends only on the previous state in the chain.*
2. *The time between state changes is a random variable T with a memoryless distribution, i.e. $\mathbb{P}(T > t + \tau \mid T > t) = \mathbb{P}(T > \tau)$ for all $t, \tau > 0$.*

A Q-matrix on a countable set of states S defines a CTMC in the following way:

If $s \neq s'$ and $Q(s,s') > 0$, then there is a transition from s to s'. If the set $\{s' \mid Q(s,s') > 0\}$ is not a singleton, then there is a competition between the transitions originating in s. The probability that transition $s \to s'$ wins the "race" is $-\frac{Q(s,s')}{Q(s,s)}$. This defines the jump chain.

The time T for leaving a state s to another state is exponentially distributed with rate $Q(s) = -Q(s,s)$ (*the total exit rate*), i.e. $\mathbb{P}(T > t) = e^{-Q(s)\cdot t}$. The exponential distribution is well-known to enjoy the memoryless-property [21, 2.3.1]. Thus a Q-matrix defines a Continuous-Time Markov Chain:

Definition 8 (CTMC with generating matrix Q). *Let Q be a Q-matrix on a countable set of states S. Then the continuous-time random process with jump chain and state-change times as decried above is the Continuous-Time Markov Chain with generator matrix Q.*

Let Q be a Q-matrix on S and Q' be a Q-matrix on S'. We call the CTMCs generated by Q and Q' *isomorphic* if there is a bijective mapping $\phi : S \to S'$ such that $Q'(\phi(s), \phi(t)) = Q(s,t)$ for all states $s, t \in S$. The transition probability matrix $P(t) = (P_{ss'}(t))_{s,s' \in S}$ describes the dynamic behaviour. It is the minimal non-negative solution of the equation

$$P'(t) = QP(t), \quad P(0) = I.$$

The (s,s')-indexed entry of $P(t)$ specifies the probability that the system is in state s' after time t if it is in state s at present. Given an initial distribution $\pi(0)$, the *transient solution* $\pi(t) = (\pi_s(t))_{s \in S}$ is then

$$\pi(t) = \pi(0)P(t).$$

In the finite case, $P(t)$ can be computed by the matrix exponential function, $P(t) = e^{Qt}$, but the numerical behaviour of the matrix exponential series is rather unsatisfactory [25]. Apart from the transient solution, which specifies the behaviour as time evolves, the *steady state* or *invariant distribution* is of great interest. It is a distribution, i.e. a map $\pi : S \to [0,1]$ with $\sum_{s \in S} \pi_s = 1$, such that $\pi Q = 0$ holds. The steady state gives information about the long term behaviour of the Markov Chain.

3.2 Stochastic Graph Transformation Systems

A stochastic graph transformation system associates with each rule name a positive real number representing the rate of the exponentially distributed delay of its application.

Definition 9 (stochastic GTS). *A stochastic graph transformation system $\mathcal{SG} = \langle TG, P, \rho \rangle$ consists of a graph transformation system $\langle TG, P \rangle$ and a function $\rho : P \to \mathbb{R}^+$ associating with every rule its application rate $\rho(p)$.*

For the rules of our sample systems $G_{random}, \mathcal{G}_{smart}$, fixed rates shall be given by $\rho(new) = \rho(kill) = 1$ and $\rho(disconnected) = 0$, while the rates of $random, smart$ shall range over 10^x for $x = 1 \ldots 4$. That means, *disconnected* shall never actually be applied, while the frequency of applying the rules for creating shortcuts will vary considerably between the experiments. This will allow us to answer the question if and under which conditions the protocol proposed in [16] is superior to a random addition of links.

Next we show how a stochastic graph transformation system gives rise to a Markov Chain, so that the analysis techniques described in Sect. 3.1 can be applied.

Definition 10 (induced Markov chain). *Let $\mathcal{SG} = \langle TG, P, \rho \rangle$ be a stochastic graph transformation system with start graph G_0 and let the induced labelled transition graph $LTG(\mathcal{G}, G_0) = \langle L, S, T, pre, post, lab \rangle$ be finitely-branching. Assume for all $s \in S$ that $\rho(p) = 0$ if $p \in R(s, s)$.*

Then the Q-matrix on S, generating the induced Markov chain of $\mathcal{SG}$ *is defined by*

$$Q(s, s') = \begin{cases} \sum\limits_{s \overset{p}{\Longrightarrow} s'} \rho(p) & , \text{ for } \quad s \neq s' \\ -\sum\limits_{t \neq s} Q(s, t) & , \text{ for } \quad s = s'. \end{cases}$$

The initial distribution $\pi(0)$ is given by $\pi_s(0) = 1$ for $s = [G_0]$ and $\pi_s(0) = 0$ else. For a proof that this is well-defined, see [11].

Note that there may be multiple transitions linking two given states. As the Q-matrix can hold only a single entry for every pair of states, the rates of all these transitions have to be added up. Hence our notion of equality on transitions determines the rate in the Q-matrix. We regard two transitions as equal only if the same rule is applied at the same match. For example, if two different peers can decide to terminate themselves, these decisions should be independent, lead to two different transitions, and finally add up to a higher rate.

3.3 Stochastic Temporal Logic

We use extended Continuous Stochastic Logic **CSL** as presented in [3] to describe properties of CTMCs. Suppose that a labelling function $L : S \to 2^{AP}$ is given, associating to every state s the set of atomic propositions $L(s) \subseteq AP$ that are valid in s. The syntax of **CSL** is:

$$\Phi ::= tt \mid a \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{S}_{\lhd p}(\Phi) \mid \mathcal{P}_{\lhd p}(\Phi_1 \mathcal{U}^I \Phi_2)$$

where $\lhd \in \{\leq, \geq\}$, $p \in [0, 1]$, $a \in AP$ and $I \subseteq \mathbf{R}$ is an interval. The other boolean connectives are defined as usual, i.e., $ff = \neg tt$, $\Phi \vee \Psi = \neg(\neg\Phi \wedge \neg\Psi)$ and $\Phi \to \Psi = \neg\Phi \wedge \Psi$. The steady-state operator $\mathcal{S}_{\lhd p}(\Phi)$ asserts that the steady-state probability of the formula Φ meets the bound $\lhd p$. The operator $\mathcal{P}_{\lhd p}(\Phi_1 \mathcal{U}^I \Phi_2)$

asserts that the probability measure of the paths satisfying $\Phi_1 \mathcal{U}^I \Phi_2$ meets the bound $\triangleleft p$.[1]

For example, the formula $\mathcal{P}_{\geq 0.02}(true\ \mathcal{U}^{[0,10]}\ disconnected)$ expresses the fact that the probability of reaching a state labelled *disconnected* within 10 time units is at most 0.02, while $\mathcal{S}_{\leq 0.01}(disconnected)$ that the long-term probability of being in a state labelled *disconnected* is less than 0.01. Both operators are also available as queries, asking for the probability of a certain formula to be true. For example, $\mathcal{S}_{=?}(disconnected)$ would return the probability of being in a *disconnected*-labelled state, rather than true or false.

In order to use **CSL** for analysing stochastic graph transformation systems, we have to define the atomic propositions AP and the labelling function L.

Definition 11 (interpreting CSL over labelled transition graphs). *Let $LTG = \langle L, S, T, pre, post \rangle$ be the labelled transition graph of a (stochastic) graph transformation system $\mathcal{G}$ with initial graph G_0. We define $AP = L$ to be the set of transition labels (rule names of $\mathcal{G}$), and the labelling of states*

$$L(s) = \{p \in AP \mid \exists t : pre(t) = s\}$$

to be given by the sets of labels of outgoing transitions.

Thus we can reason about the applicability of rules. Coming back to the above example, a state labelled *disconnected* is therefore one where the rule *disconnected* is applicable (which has an outgoing transition with that label). $\mathcal{S}_{=?}(disconnected)$ therefore queries the transition system for the probability of being in a disconnected state.

Recall that rule *disconnected* does not have any effect on the state, i.e., it is exclusively used to represent a state property. The transition rates of such property rules are set to 0, so that they do not affect the Q-matrix.

4 Application

We have constructed an experimental tool chain consisting of GROOVE [23] for generating the labelled transition graph of a graph transformation system, and PRISM [14] for probabilistic model checking. An adapter connects both tools by translating the transition graph generated by GROOVE into a PRISM transition system specification, incorporating the transition rates ρ as specified in a separate file [2].

As usual, the size of the state space to be generated and analysed is a limiting factor. Presently the main bottleneck is not the actual state space generation in GROOVE, which can handle up to 10^6 states, but its import into the PRISM model checker, which reaches its limits at a few thousand states. The actual model checking, once the model is successfully imported, takes no more than a few seconds.

[1] The other path and state operators can be derived. Details are given in [3].

[2] `http://www.ls10.de/sgt`

The problem is caused by the low-level presentation of transition systems generated by the transformation tool, which uses a single state variable s only. Transitions are represented as conditional assignments as in the listing below, where `[new]s=176->1*new_rate:(s'=80)` defines a transition from state 176 to state 80 using rule *new* at rate $new\_rate = 1$. The enumeration at the end defines the labelling of states by atomic propositions (= rule names).

```
stochastic
// 605 Nodes
// 14322 Transitions
const int kill_rate=1;
const int smart_rate=1000;
const int new_rate=1;
const int disconnected_rate=0;
module M s :  [0..604] init 438;
    [new] s=176 -> 1*new_rate:(s'=80);
    [kill] s=359 -> 2*kill_rate:(s'=537);
    ...
    [disconnected] s=101 -> 4*disconnected_rate:(s'=422);
endmodule
// label "smart" = (s= 227, 159, 587, 247, 194);
// label "kill" = (s= 359, 174, 202, 151, 264, 126, ...);
// label "new" = (s= 176, 341, 324, ...);
// label "disconnected" = (s= 95, 364, 302, 116, 402, ...);
```

The limitation in the number of states requires a style of specification where all operations are specified by single rules, rather than breaking them down into smaller steps. The latter would lead to simpler rules, but create intermediate states. The use of path expressions and application conditions is essential for this style of specification.

The results of applying the tool chain to the two stochastic graph transformation systems defined in the previous section are visualised in Fig. 7.

Both systems have been restricted to a maximum of 7 peers and one registry. The bottom graph represents the behaviour of $\mathcal{G}_{smart}$ whose transition graph has 798 states and 16293 transitions.

We observe that, increasing the rate of rule *smart* by a factor of 10 we decrease the long-term probability for a disconnected network by about the same factor: from 0.225300 for $\rho(smart) = 10$ to 0.000244 for $\rho(smart) = 10000$. Indeed, for rates at least 10 times higher than those of *kill* and *new*, the probability seems to go against $2.4 \cdot \rho(smart)^{-1}$. That means, an estimate of the average time it takes to execute (the implementation of) *smart* in relation with the rate of peers entering and leaving the system would provide us with an estimate of the networks reliability.

The upper graph in Fig. 7 represents the system $\mathcal{G}_{random}$ which has 487 states and 9593 transitions. We observe that the added redundancy does not have a relevant effect on the reliability, even if the number of additional edges created is roughly the same as in the other system (the overall number of states is only slightly smaller). This shows the superiority of the first system.

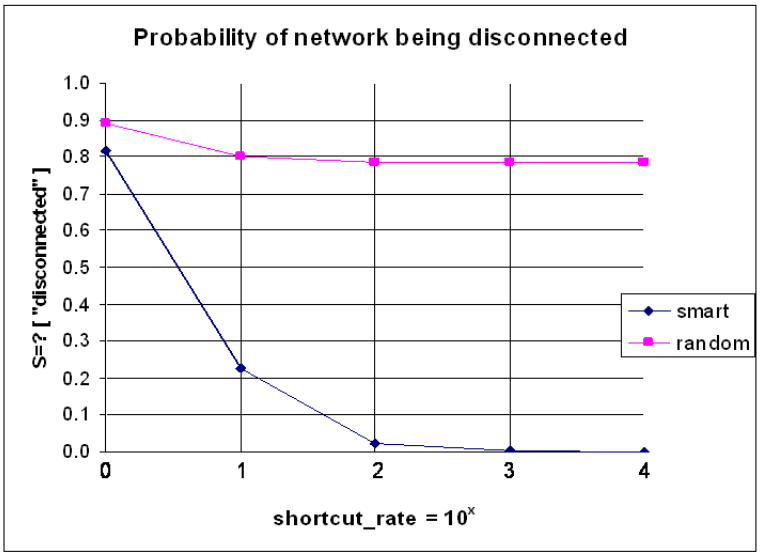

Fig. 7. Results of stochastic model checking

5 Conclusion

In this paper we have developed a case study in stochastic graph transformation to validate the practicability of the approach and understand its limitations. The problem addressed, a protocol for adding redundant links in a P2P network, has been modelled and analysed using an experimental tool chain. Let us conclude this paper by discussing some of the issues and lessons learned in this exercise.

First, the model in this paper captures only a simplified version of the original protocol. A complete presentation would have required even more advanced features, like rule priorities or multi-objects, which are partly beyond the abilities of available analysis tools. Alternatively, a high amount of encoding of standard graph algorithms would have rendered the approach useless for model checking.

A possible solution to this problem is the use of procedural abstractions as provided by programming-oriented graph transformation approaches like FUJABA [26]. Ideas for structuring stochastic graph transformation systems into modules could be used to encapsulate the implementation of these procedures [12].

Second, P2P networks often contain thousands or even millions of nodes. Hence, the validity of the results of our analysis, which only considers seven peers, can be questioned. However, this is not so much an issue of the formalism itself, but of the analysis techniques and tools. We expect that much more realistic data can be obtained by complementing model checking with stochastic simulation.

Finally, it depends on the specific application domain whether user behaviour, as expressed in rules like $new, kill$, or system behaviour like in $smart, random$ is

exponentially distributed. Future work will extend the approach to allow different kinds of distributions.

Acknowledgement. The author wishes to express his gratitude to Arend Rensink for numerous new versions of GROOVE to cater for the needs of the case study, Sebastian Menge for his support with the transformation tool from GROOVE to PRISM transition systems, as well as Thomas Erlebach, Georgios Laijos, and Leonardo Mariani for comments and discussions.

References

1. M. Ajmone-Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing. John Wiley and Sons, 1995.
2. William G. Anderson. *Continuous-Time Markov Chains*. Springer, 1991.
3. Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model checking continuous-time markov chains by transient analysis. In *Computer Aided Verification*, pages 358–372. Springer, 2000.
4. Falko Bause and Pieter S. Kritzinger. *Stochastic Petri Nets*. Vieweg Verlag, 2nd edition, 2002.
5. E. Brinksma and H. Hermanns. Process algebra and Markov chains. In J.-P. Katoen E. Brinksma, H. Hermanns, editor, *FMPA 2000*, number 2090 in LNCS, pages 183–231. Springer, 2001.
6. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26(3,4):241–266, 1996.
7. P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. IPA Dissertation Series 1999-10, CTIT PhD-Thesis Series 99-25, University of Twente, November 1999.
8. H. Ehrig, M. Pfender, and H.J. Schneider. Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973.
9. T. Gilb. *Principles of Software Engineering Management*. Addison-Wesley, 1988.
10. A. Habel, R. Heckel, and G. Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, 26(3,4):287 – 313, 1996.
11. R. Heckel, G. Lajios, and S. Menge. Stochastic graph transformation systems. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *Proc. 2nd Intl. Conference on Graph Transformation (ICGT'04), Rome, Italy*, volume 3256 of *LNCS*, pages 210 – 225. Springer-Verlag, October 2004.
12. R. Heckel, G. Lajios, and S. Menge. Modulare Analyse Stochastischer Graphtransformationssysteme. In P. Liggesmeyer, K. Pohl, and M. Goedicke, editors, *Software Engineering 2005, Essen, Germany*, volume 64 of *Lecture Notes in Informatics*, pages 141 – 152. GI, March 2005.
13. M. Korff and L. Ribeiro. Concurrent derivations as single pushout graph grammar processes. In *Proc. Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation (SEGRAGRA)*, volume 2 of *Electronic Notes in TCS*, pages 113–122. Elsevier Science, 1995.
14. M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In T. Field, P. Harrison, J. Bradley, and U. Harder, editors, *Proc. 12th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002.

15. M. Löwe. Algebraic approach to single-pushout graph transformation. *Theoret. Comput. Sci.*, 109:181–224, 1993.
16. L. Mariani. Fault-tolerant routing for p2p systems with unstructured topology. In *Proc. International Symposium on Applications and the Internet (SAINT 2005)*, Trento (Italy), 2005. IEEE Computer Society.
17. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoret. Comput. Sci.*, 96:73–155, 1992.
18. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
19. M. K. Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. PhD thesis, University of California, 1981.
20. S. Natkin. *Les Réseaux de Petri Stochastiques et leur Application à l'Evaluation des Systèmes Informatiques*. PhD thesis, CNAM Paris, 1980.
21. James R. Norris. *Markov Chains*. Cambridge University Press, 1997.
22. C. Priami. Stochastic π-calculus. *The Computer Journal*, 38:578 – 589, 1995. Proc. PAPM '95.
23. A. Rensink. The GROOVE simulator: A tool for state space generation. In J.L. Pfaltz, M. Nagl, and B. Böhlen, editors, *Applications of Graph Transformation with Industrial Relevance Proc. 2nd Intl. Workshop AGTIVE'03, Charlottesville, USA, 2003*, volume 3062 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2004.
24. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*. World Scientific, 1997.
25. W. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.
26. University of Paderborn Software Engineering Group. The Fujaba Tool Suite. `www.fujaba.de`.