Electronic Communications of the EASST
Volume 16 (2009)

Proceedings of the Doctoral Symposium at the International Conference on Graph Transformation (ICGT 2008)

Model-based Simulation of VoIP Network Reconfigurations using Graph Transformation Systems

Ajab Khan, Paolo Torrini, Reiko Heckel

20 pages

Guest Editors: Andrea Corradini, Emilio Tuosto
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Model-based Simulation of VoIP Network Reconfigurations using Graph Transformation Systems

Ajab Khan, Paolo Torrini, Reiko Heckel

University of Leicester, {ak271,pt95,reiko}@mcs.le.ac.uk

Abstract: We address the modelling and validation of P2P networks with special attention for problems related to VoIP services, focusing particularly on Skype. We use generalised stochastic graph transformation systems and associated stochastic simulation techniques based on generalised semi-Markov processes.

Keywords: Graph Transformation, Stochastic Simulation, Voice over IP

1 Introduction

Today's Internet is already used as a public switch telephone network (PSTN), based on the concept of Peer to Peer (P2P) to share voice, video and data. Among several VoIP applications Skype is the most popular [AKK07]. Voice services are the prime source of revenue for carrier service providers — though margins are low and dropping. Carrier service providers can reduce operational costs and increase revenues by shifting from Time Division Multiplexing (TDM) to IP based interconnection. However, concerns about Quality of Service (QoS) have caused many carriers to postpone technology platform shifts [Tel06].

In P2P networks there is no centralised control. The transfer of data, voice and video takes the form of a flow through intermediate nodes, but the nodes are free to join and leave the network, and they are allowed to behave selfishly by blocking the routing of traffic for third parties. This may result in a need for frequent architectural reconfigurations [GS04]. In the case of VoIP traffic, the network has to recover particularly fast, so that the quality of service is not affected [AKK07].

P2P VoIP traffic potentially suffers from performance issues like packet loss, communication delay, jitter and echo [Glo06], which can greatly affect QoS. Packet loss and communication delay can occur principally due to network reconfigurations associated with peer dynamics. Jitter can result from packets arriving at variable time intervals, due either to network congestion, reconfiguration or peer dynamics.

Various solutions have been proposed to this sort of problems, e.g., [GS04] proposes that an incentive should be given to intermediate nodes and resource owners, [Hec05] proposes to maintain redundant links between peers, [LMP04] proposes an approach based on changes in routing strategies. The peer dynamics and complexity of the P2P network can make it hard and quite expensive to validate these solutions through classical means [Hec05, MSZ03].

We propose to use model-based simulation to study reconfiguration in P2P networks. The aim is to model protocols for evaluating and improving the QoS properties of VoIP

applications. We consider the P2P network architecture as a graph, in which network nodes are represented by graph vertices and graph edges represent network connections. Reconfiguration in such a network can be modelled by means of graph transformation [Hec05]. Stochastic analysis techniques can be used for validation.

2 Generalised Stochastic Graph Transformation

In this section, we provide the basic notions of typed and stochastic graph transformation together with their semantic model of generalised semi-Markov processes and the probability distributions needed for our case study. Following a different approach from that presented in [KL07], these notions generalise those introduced in [HLM06, Hec05] by allowing for general probability distributions.

2.1 Graph Transformation Systems

Given a type graph TG, a *typed graph* (over TG) is a pair $\langle G, g \rangle$ of a graph G and a graph morphism $g : G \to TG$ that assigns types to nodes and edges [EEPT06]. We are assuming that nodes can carry attributes. A graph transformation rule consists of an injective partial graph morphism $r : L \to R$. A match for $r : L \to R$ into some graph G is a total injective graph morphism $m : L \to G$. Given a rule r and a match m for r in a graph G, we say that $\langle r, m \rangle$ is a rule match in G. The SPO transformation from G with r at m, denoted by $G \Rightarrow_{r,m} H$ or simply $G \Rightarrow_r H$, is defined by the pushout of r and m in the category of graphs and partial graph morphisms.

$$\begin{array}{ccc} L & \xrightarrow{r} & R \\ {\scriptstyle m}\downarrow & & \downarrow{\scriptstyle m^*} \\ G & \xrightarrow{r^*} & H \end{array}$$

For example, the rule *deliver packet* in fig. 7(d) has a left-hand side with three typed nodes (p,sn and $sc2$) and three edges (of type *receiver* from p to $sc2$, *link* from sn to $sc2$, and *at* from p to sn). Its application removes the *at* edge and introduces a new one between p and $sc2$, as shown in the right-hand side. The first two graphs in fig. 8 exemplify how this rule can be applied — the left-hand side of the rule is mapped to the graph on the left by matching nodes p and $p1$, sn and sn, $sc2$ and $sc2$, and so the graph on the right is obtained.

A graph transformation system (GTS) $\mathcal{G} = \langle TG, P, \pi, G_0 \rangle$ consists [KL07] of a type graph TG, a set P of rule names, a function mapping each rule name p to a TG-typed rule $\pi(p) = r : L \to R$, and an initial TG-typed graph G_0. We make use of negative application conditions (NACs) [EEPT06], shown as crossed-out nodes and edges in the rules' left-hand sides.

A *numbered graph* is a graph whose sets of nodes and edges form subsets of the natural numbers. To capture the idea of a concrete and deterministic implementation of SPO graph transformation, we assume a choice Σ of direct transformations such that

- for every rule $r : L \to R$ and match m for r in a graph G there is a unique result of applying r at m, denoted by $\Sigma(G, r, m) = (H, r^*, m^*)$;
- names are chosen consistently in consecutive transformations, preserving the identities of nodes and edges where possible, and never reusing names from the past, i.e., for every sequence of transformations $G_0 \Rightarrow_{r_1} G_1 \Rightarrow_{r_2} \cdots \Rightarrow_{r_n} G_n$ and $0 < i < j \leq n$ it holds that
  1. $x \in dom(r_j^* \circ \cdots \circ r_i^*)$ implies $r_j^* \circ \cdots \circ r_i^*(x) = x$
  2. $x \in G_i \cap G_j$ implies $x \in dom(r_j^* \circ \cdots \circ r_i^*)$;

Limiting ourselves to graphs over natural numbers, the collection of all finite numbered graphs is actually a set — whereas in general the collection of *all* finite graphs is a proper class. The requirement about the consistency of names along sequences of transformations is easily satisfied if we choose fresh names for all new elements and never reuse names of elements that are deleted.

For a GTS $\mathcal{G} = \langle TG, P, \pi, G_0 \rangle$, we denote by $\mathcal{R}_\mathcal{G}$ the set of the graphs that are reachable, and we denote by $\mathcal{M}_\mathcal{G}$ the set of all matches $m : L \to G$ for rules $r : L \to R$ in $\mathcal{G}$ into TG-typed numbered graphs $G \in \mathcal{R}_\mathcal{G}$. We define a relation $\equiv$ over matches, as follows: for all matches $m : L \to G, n : L \to H \in \mathcal{M}_\mathcal{G}$ for the same rule $r : L \to R$, $m \equiv n$ iff for all nodes and edges x of L, $m(x) = n(x)$. It is straightforward to see that this is an equivalence relation. We define the *event set* of $\mathcal{G}$ to be the set of pairs

$E_\mathcal{G} = \{\langle p, [m] \rangle | p \in P \wedge \pi(p) : L \to R \wedge [m : L \to G] \in \mathcal{M}_\mathcal{G} / \equiv\}$ satisfying ps application conditions

Broadly speaking, events can be associated to equivalence classes of rule matches, and thus outlive the target graph of the match if the difference does not affect the elements in the codomain.

2.2 Generalised stochastic graph transformation systems

In order to reason stochastically about GTSs, we want to associate with each event a distribution function governing the execution of the application of the corresponding step.

We say that $\mathcal{S}_\mathcal{G} = \langle \mathcal{G}, F \rangle$ is a *generalised stochastic graph transformation system* whenever $\mathcal{G}$ is a GTS and $F : E_\mathcal{G} \to (\mathbb{R} \to [0,1])$ is a function which associates with every event in $\mathcal{G}$ a continuous distribution function. We assume $F(e)(0) = 0$ (*null delay* condition) [KL07].

This definition in a sense generalises a previous proposals presented in [KL07] by making the probability distribution dependent on the event (rule name and match) rather than just on the rule name — although here we have a restriction to numbered graphs that they have not. Stochastic graph transformation systems (SGTS) as introduced in [HLM06] on the other hand allow only for exponential distributions associated with rule names.

Our interest in stochastic graph transformation systems is closely associated with simulation, where the stochastic aspect is useful in order to resolve the non-deterministic

character of ordinary GTSs. This also motivates our interest in numbered graphs, reflecting the handling of names which is typically associated to an implementation of the approach in an object-oriented language.

We rely on standard notions of stochastic process and discrete event system [CL08] for an intuitive presentation of our approach. The behaviour of a stochastic GTS can be described as a stochastic process over continuous time, where reachable graphs form a discrete state space, the application of transformation rules defines state transitions as instantaneous events, and interevent times, determined by the application of transformation rules, are dominated by continuous probability distributions. More precisely, we associate each rule, as it becomes enabled by a match, with an independent random variable (*timer*) which represents the time expected to elapse (*scheduled delay*) before the rule is applied to the match. This timer is set randomly, based on a continuous probability distribution function. The null delay condition means that the probability of a null delay at enabling time is always zero.

In general, assuming time is continuous and events are instantaneous makes it possible to rule out simultaneous events in the model. More precisely, the fact that the distributions are continuous, the assumption that events are instantaneous and that timers are independent variables, together with the limit condition $F(e)(0) = 0$, suffice to guarantee that the probability that two rules are applied simultaneously is always zero. This fact also guarantees that the implicit "race condition" in the application of transformation rules makes it possible to resolve non-determinism without the need to allow for explicit parallelism.

2.3 Generalised semi-Markov processes

Generalised semi-Markov processes (GSMP) can be particularly useful in modelling the behaviour of GSGTSs [KL07]. GSMPs are stochastic processes generated by structures (generalised semi-Markov schemes) that can be regarded as a generalisation of continuous-time Markov chains, just as Markov processes are generated by Markov chains. GSMPs can also be regarded as extensions of Markov chains with timers associated to transitions between states [Nel95].

Markov processes enjoy the memoryless property. This means that at each time, which transition is going to fire depends only on the current state — in other words, it is independent from the past states as well as from the time spent in the current one. No matter how long the system has been in a state, the probability of experiencing a further delay remains the same. This property translates into the fact that, when a Markov process is generated by a set of timed transitions, timers associated with transitions, and consequently also interevent times, must be exponentially distributed.

In semi-Markov processes, timers as well as the resulting interevent times can be generally distributed. This corresponds to a model where transitions are generally independent of the past states, but depend on their timers which may have been set in previous states [DK05]. More formally, GSMPs can be defined as the processes generated by specific stochastic structures called generalised semi-Markov schemes (GSMS) [DK05]. A GSMS is a structure

$$\begin{aligned}\mathcal{P} = \langle\ & Z \\ & E \\ & \mathit{active} : Z \to \wp E \\ & \mathit{new} : Z \times E \to Z \\ & \Delta : E \to (\mathbb{R} \to [1,0]) \\ & s_0 : Z \qquad \rangle\end{aligned}$$

where Z is a set of system states; E is a set of implicitly timed events; *active* is the activation function, so that $\mathit{active}(s)$ is the finite set of active events associated with s; *new* is the transition function depending on states and events; Δ is the distribution assignment, so that $\Delta(e)$ is the probability distribution function associated with the scheduled delay of event e; and s_0 is the initial state.

2.4 Time and probability distributions

An important aspect in the modelling of a real-time system is the representation of their time, as distinct from the time of the simulation. The representation of real time used in our model is an asynchronous one, following the approach presented for GTSs in [GVH03], and it is obtained by means of *chronos* attributes — essentially, one for each component associated with a clock in the system (the nodes of type *Node)* — see section 3), and of a *chronos* rule — meant to be the sole modifier for each *chronos* attribute, matched by each clocked node, and therefore always enabled by each of them. In a GSGTS each *chronos* attribute of a *Node* graph component can represent the physical clock associated with the corresponding system component. Each application of the *chronos* rule represents a tick of the matching clock. Clocks are stochastically synchronised with the time of the simulation by means of the normally distributed timers associated with each *chronos* rule match. This approach has the advantage of being quite flexible and very close to the actual measurement of time in the real system.

Chronos rule matches are indeed a good example of why exponential distributions could not suffice in our case. Concede that a timer is exponentially distributed — then it can be reset every time the state changes, i.e., every time any rule is applied. This can fit with modelling instant reactions, but hardly goes well with a description of physical clocks. In contrast, when a *chronos* rule match is enabled and the corresponding timer is set, we do not want it to be reset at each state change before the rule can be applied. Moreover, the memoryless property does not help to limit the presence of isolated cases with big deviations from the mean — something at odds with the behaviour of a functioning clock. Exponential distributions tie together mean and variance, respectively $1/\lambda$ and $1/\lambda^2$ where λ is the exponential rate — so it is possible to decrease standard deviation only by decreasing the expected mean. What we need for *chronos* rule matches, on the other hand, are distributions that take into consideration

the time interval a rule has been enabled for. In normal distributions mean and variance are distinct parameters, allowing for a finer modelling. Moreover, normal distributions have a quite stable character, intuitively visualised by the bell curve functions associated with them — meaning that essentially, experimental values tend to hurdle within short range from the mean in terms of standard deviation; correspondingly, excess kurtosis is 0 (few big deviations, many small ones), and so is skewness (little asymmetry in deviations) [Nel95, BA07].

As an example of graph transformation rule in a GSGTS we then consider the *chronos* rule *clock tick* in fig. 2. This rule can be applied to a node $n : Node$ representing a component of the system that has a clock with time set as value of the *chronos* attribute. Each application of the rule increases that value by one. The simulation time corresponding to a clock tick can be randomly determined depending on the probability distribution function associated with the rule match.

2.5 Translation of GSGTSs and CGSGTSs into GSMPs

The translation of a GSGTS into a GSMP described in [KL07] is based on the construction of the unfolding grammar associated with the underlying GTS. The unfolding of a GTS provides essentially a global name space — obtained as a pushout construction over all the finite prefixes of the GTS, where each finite prefix is a model obtained from the initial graph by applying the rules in every possible way, up to a finite causal depth, without deleting the left hand-sides, but rather preserving all the information about each rule application [BCM99].

The more concrete representation in this paper, motivated by our interest in simulation using existing graph transformation tools, replaces the global name space provided by the unfolding by an ad-hoc definition of a name space of numbered graphs and the assumption that fresh names are used whenever elements are created, names are never reused, and names are preserved whenever possible.

Now, given a GSGTS $\mathcal{S}_{\mathcal{G}} = \langle TG, P, \pi, G_0, F \rangle$, we can define its translation as a GSMP

$$\mathcal{P} = \langle \mathcal{R}_{\mathcal{G}}, E_{\mathcal{G}}, active, new, \Delta, G_0 \rangle$$

where the set of states $\mathcal{R}_{\mathcal{G}}$ is the subset of TG-typed numbered graphs reachable in $\mathcal{G}$; $E_{\mathcal{G}}$ is the set of events (rule - match pairs) in $\mathcal{G}$; $active(G)$ is the set of the events $\langle p, [m] \rangle$ such that m is a match for $p \in P$ in G; the transition function is defined with $new(G, \langle p, [m] \rangle) = H$ whenever $r = \pi(p)$ and $\Sigma(G, r, m) = (H, r^*, m^*)$; $\Delta(\langle p, [m] \rangle)$ is defined as $F(\langle p, [m] \rangle)$; and G_0 is the initial typed graph in $\mathcal{G}$.

3 Case Study: Skype Network Architecture

The most popular VoIP application is Skype. It has more than 50 million users — an increasing figure, and it is based on a P2P architecture. The Skype network offers three services: VoIP, instant messaging and file transfer [GDJ06, BS06]. The Skype architecture is described by Figure 1. Skype nodes are distinguished into clients (SC) and super nodes

(SN) [XY07]. SNs maintain an overlay network. SCs have to connect to one of the SNs, which act as telephone switches and routers for their clients. The Skype network is subject to architectural reconfiguration whenever a new SC joins the Network, an SN leaves the network, an intermediate SN fails to route the traffic or has connectivity problems.

In our model we assume that, after registration, each Skype client keeps a permanent connection with an SN node: first the SC tests the latency of an arbitrary SN node and if the latency is in the range of the standard, i.e. 800 ms [Lin07], it establishes a connection. Alternatively, we can let the SCs choose their connection based on the spare bandwidth and/or memory of the available SNs.

Besides latency, jitter is another important aspect and it plays a vital role in maintaining the Quality of Service (QoS) in the VoIP traffic. In our model we consider that when the link is established for communication between two SC nodes, the receiver SC computes the latency of the arriving packets by time-stamping the packets. If the difference between two arriving packets is more then 40ms [Tel06] the network is reconfigured and jitter is calculated afresh for a new route [Spi07].

The model can be best described with the help of a type graph (See Figure 1), including registration servers (RS), super-nodes (SN) Skype clients (SC), packets, a type *Node* that generalises both SC and SN, and edge types *link*, *registration*, *overlay*, *sender* and *receiver*.

The modelling of time follows the approach of a unique time stamp attribute *chronos* [GVH03] associated with nodes of the graph, as required[Glo06]. The *chronos* rule, named *clock tick* and pictured in figure 2, is used to advance time for each node of type *Node*, by increasing the value of the *chronos* attribute. Note that each match of this rule corresponds intuitively to a specific clock associated with a specific distribution.

4 Skype Reconfiguration as Graph Transformation

We will now introduce a set of rules based on the following scenario that we would like to model. When a client tries to establish a network connection, it has to get registered with the central registration server. The client may receive the addresses of several super-nodes, and it can make a choice between them based on their reachability, available bandwidth, etc.. Before establishing the connection with a nearby super-node, the client can work out the communication delay (latency) with respect to it. In order to do that, the client sends out a packet to the super-node and waits for it to be sent back. The client can then compute the latency by comparing the time stamps.

Rules in Figure 3: create, send and return time-stamped packet. The first rule (*a*) creates a packet *p* at SC and sets the *chronos* attribute of *p* to the current time of the client node. Note that *p.chronos* is just a static time stamp, as packets are not of type *Node* and so *clock tick* cannot be applied to them. Rule *b* sends the packet to the super-node. Rule *c* returns the packet to the client. The packets are used to find out the time delay between the client and the super-node. Comparing time stamps allows the peer to connect to a super-node which offers the lowest latency as per ITU-T. If the delay is in the range of the standards of ITU-T then the link with the super-node is established.

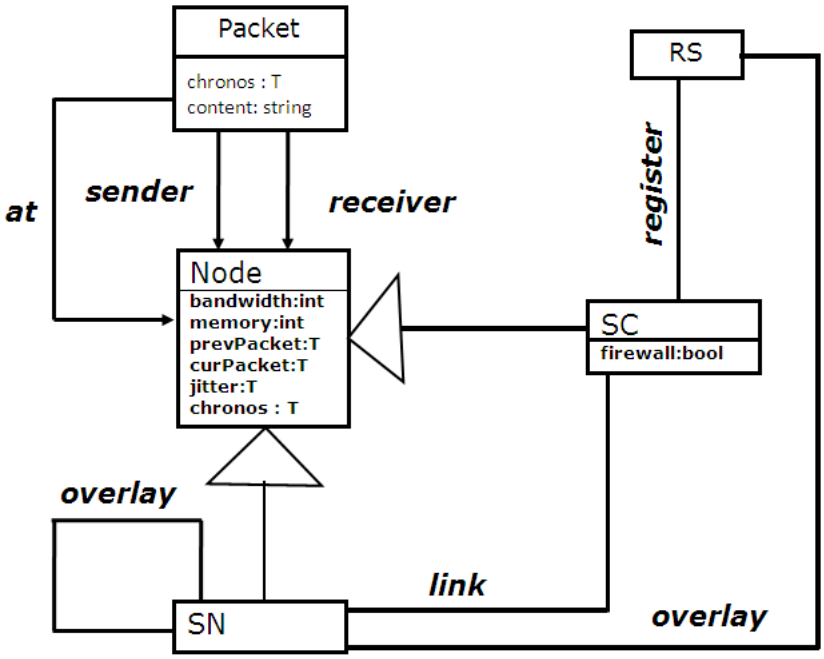


Figure 1: Type graph

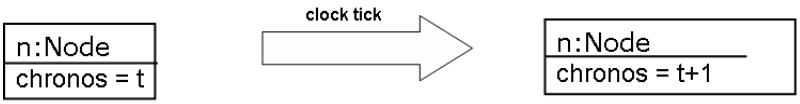


Figure 2: Clock tick

Rules in Figure 4: connect and reject. These rules are used to compute the latency and either effect or reject the connection. The link between SC and SN is established if the SC's current time (*sc.chronos*) at the reception of the ping *reply* has advanced no more than 800ms with respect to the time stamp on the original *ping* message (*p.chronos*). If the delay is in standard range, SC will be linked with SN, and the bandwidth of the SN will be reduced by 5Kbps [Sky06]. If greater, the current connection attempt is aborted and a new one with another super-node is tried. After the client gets connected to a super-node, it can communicate directly with other peers by finding their address from the global index which is maintained by the central login server.

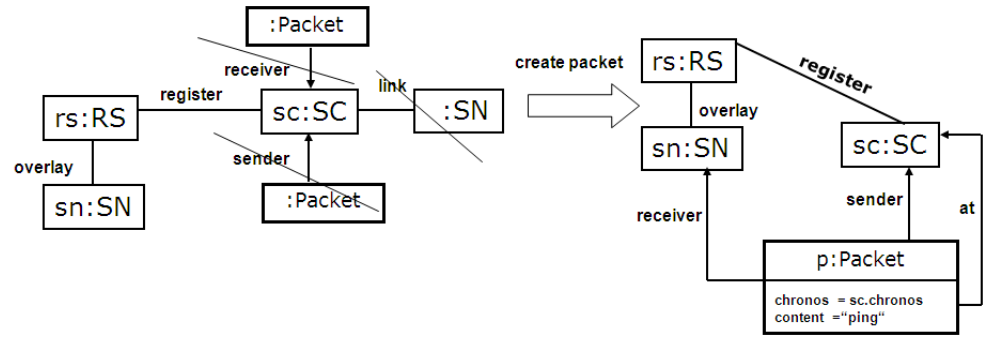
Rules in Figure 5: SN selection maximising bandwidth. A client may encounter a set of viable super-nodes. In that case the client may try to find out the super-node with the maximum available bandwidth. This is an example of a selection being made, among the possible matches for a rule, in order to maximise a certain character. Note that such a condition can be translated into a negative application condition along the lines of: *there is no other SN with higher bandwidth than the chosen one*.

Rule in Figure 6: jitter attenuation. We can model also jitter as a temporal property. There are several ways to calculate jitter [Spi07]; here we are looking into the packet inter-arrival method. We use three attributes: *curPacket* — used to keep the arrival time of the current (last) packet; *prevPacket* — to keep the arrival time of the previous packet; and *jitter* — to store the difference between the two, whenever a new packet arrives. The requirement is that inter-arrival times do not exceed the standard 40ms [Tel06]. Otherwise the connection to the current super-node is abandoned and a new connection is sought.

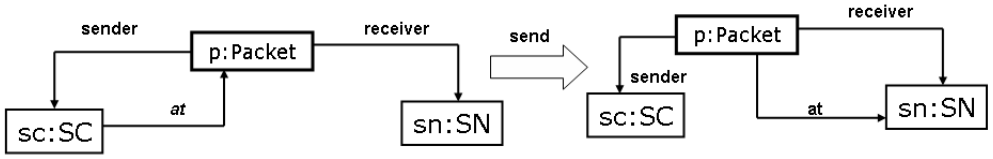
Rules in Figure 7: packet transportation rules. We model a packet as a node. In the communication between two clients we consider two cases. If both clients have public IPs and they can be located in the network (see Figure 7(a)), packets can be transferred directly, according to the P2P architecture, from sender to receiver. Alternatively either the sender or the receiver are behind firewalls, or else hidden by network address translation (NAT). In this case the sender client will first forward the packet to the super-node and then the super-node will route the packet to the receiving client.

5 Simulation

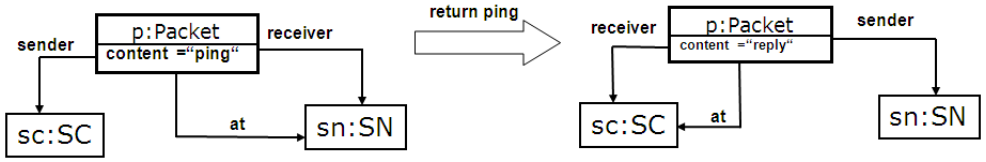
Graph transformation systems can support a variety of validation and verification techniques. Model checking based on CSL and stochastic simulation techniques based on translation to Markov chains were introduced in [HLM06] for SGTS. Model checking can be useful to formally verify abstract properties of processes, but this may turn out to be unfeasible in case of complex examples and particularly in the case of general distributions. On the other hand, Monte Carlo-style stochastic simulation is based on the execution of particular processes which are chosen probabilistically by means of a random number generator (RNG), i.e. a program that can generate pseudo-random numbers depending on a distribution function. The result of stochastic analysis is typically obtained in terms of average values after running several times the simulation. In



(a) Create time stamped *ping* packet

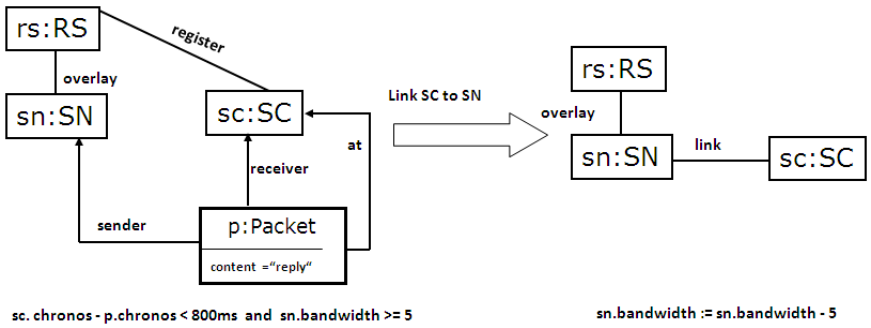


(b) Send packet

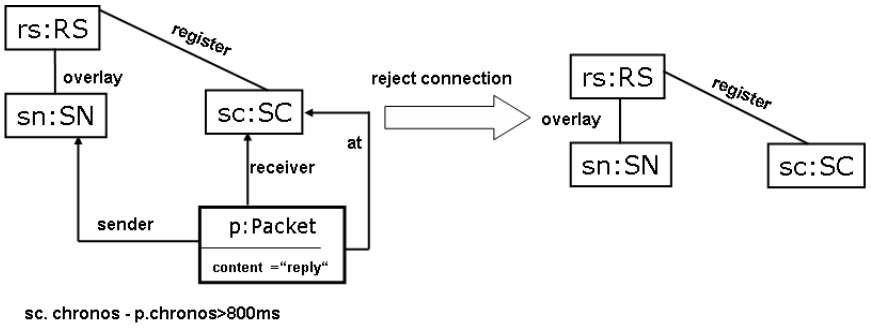


(c) Return *reply* packet

Figure 3: Determine round-trip delay



(a) Connect SC to SN



(b) Reject connection

Figure 4: Connection to/rejection of SN based on round-trip delay

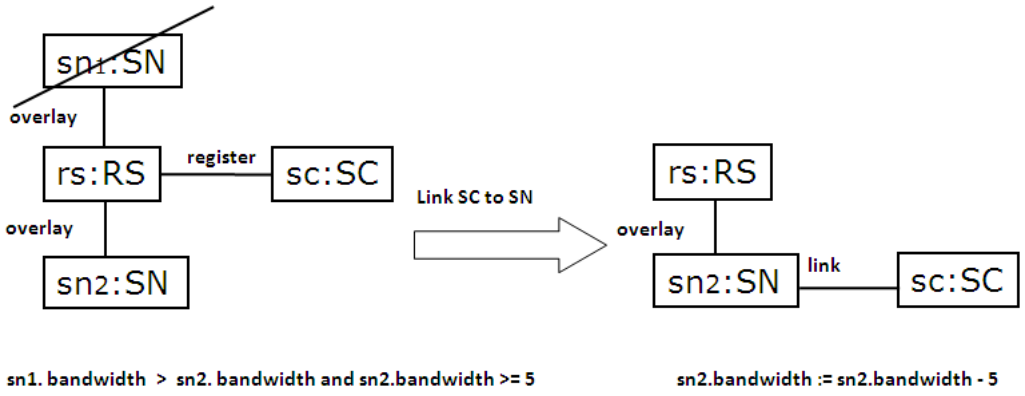


Figure 5: SN selection maximising bandwidth

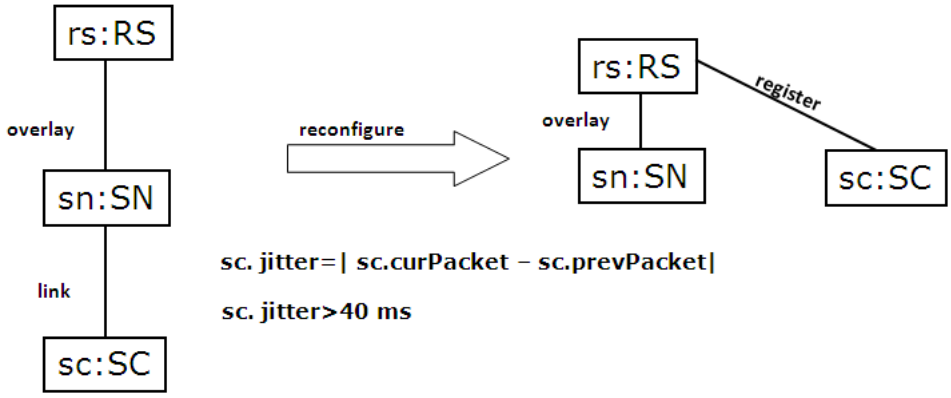


Figure 6: Jitter attenuation

the following we describe an algorithm for the stochastic simulation of a GSMP [KL07]. We then proceed to illustrate the architecture of a tool for GSGTS simulation, and present an architecture-specific refinement of the algorithm.

5.1 A GSMP algorithm

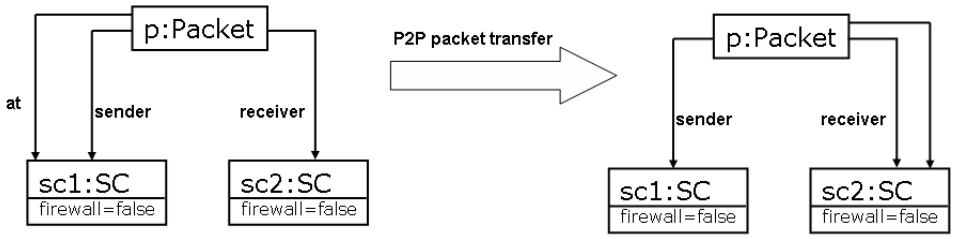
A simulation algorithm for a GSMP has been presented in [KL07] as a version of the *Event Scheduling Scheme* in [CL08]. The algorithm relies on calls to an RNG in order to provide delay values for the timers associated with newly activated events. Simulation time is recorded explicitly. Active timed events are managed as a list which is ordered by the time events are scheduled to take place at, so that the first element of the list turns out to be the event scheduled to take place first.

For the initialisation phase, given a GSMP $\mathcal{P} = \langle Z, E, active, new, \Delta, s_0 \rangle$

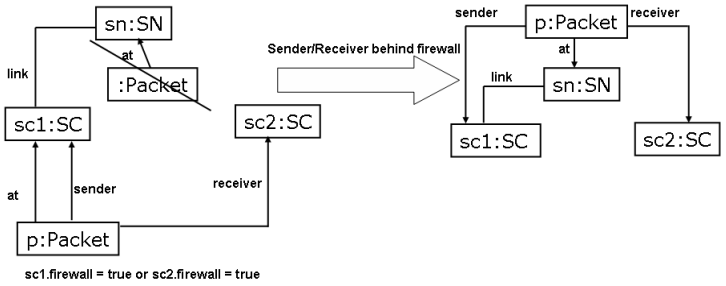
1. the simulation time is initialised at t_0.
2. The set of the activated events $A = active(s_0)$ is computed.
3. For each event $e \in A$, a scheduling time t_e is computed by adding t_0 to a random delay value d_e given by the RNG depending on the probability distribution function $\Delta(e)$;
4. The active events with their scheduling times are collected in the scheduled event list $l_{s_0} = \{(e, t_e) | e \in A\}$, ordered by the time values.

Then, for each simulation step, given the current state $s \in S$ and the associated scheduled event list $l_s = \{(e,t) | e \in active(s)\}$

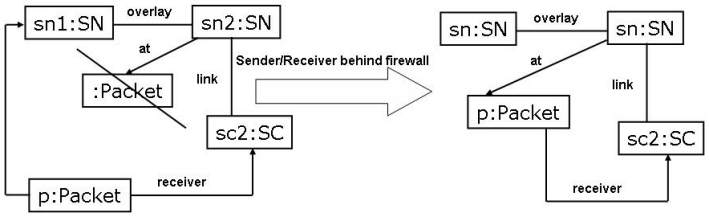
1. the first element $k = (e,t)$ is removed from l_s;
2. the simulation time t_S is updated by increasing it to t;



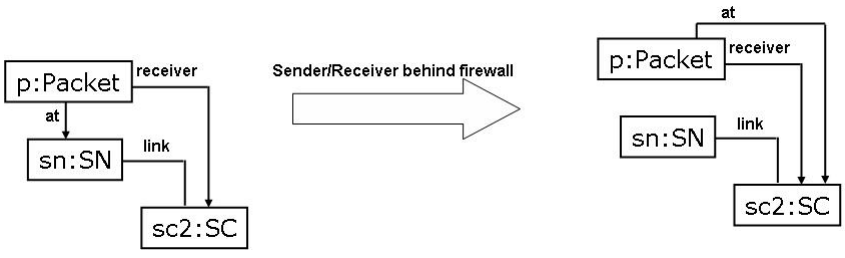
(a) P2P packet transportation



(b) Send packet to sender host SN



(c) Send packet to receiver host SN



(d) Deliver packet to receiver

Figure 7: Packet transportation rules

3. the new state s' is computed as $s' = new(s,e)$;

4. the list $m_{s'}$ of the surviving events is computed, by removing from l_s all the elements become inactive, i.e. all the elements (z,x) of l_s such that $z \notin active(s')$;

5. a list $n_{s'}$ of the newly activated events is built, containing a single element (z,t) for each event z such that $z \in active(s') \backslash active(s)$ and has scheduling time $t = t_S + d_z$, where d_z is a random delay value given by the RNG depending on the distribution function $\Delta(z)$;

6. the new scheduled event list $l_{s'}$ is obtained by reordering the concatenation of $m_{s'}$ and $n_{s'}$ with respect to the time values.

5.2 A tool for GSGTS analysis

In the case of a GSMP obtained as translation of a GSGTS, a potential state space explosion problem may arise from the computation of *active*, i.e. of the rule matches that are active in the current graph. Incremental pattern matching [VV04] is an approach in the implementation of graph transformation systems that can help to cope with this issue. Instead of computing all the matches for each state, a tree of partial matches is built from the initial graph in a pre-processing phase, and this structure gets subsequently updated at each transformation by adding the new matches and deleting those that become disabled. This approach has been implemented in Viatra [BHRV08], a graph transformation tool which can support attributes and negative conditions, available as an *Eclipse* plug-in.

Essentially, what we need to do is to integrate a graph transformation tool (GTT) with an implementation of stochastic simulation based on GSMPs. We expect to implement the stochastic simulation tool (SST) in Java, relying on SSJ [YCSC02] for the stochastic libraries and random number generation.

5.3 A GSGTS algorithm

The following gives the pseudo-code for a GSGTS simulation algorithm that is essentially a refinement of the Event Scheduling Scheme considered in section 5.1. The code is loosely functional in style [Tur92], allowing for imperative features — commands (typed with IO), references (singled out with #), as well as for a component-based structure to make clearer the general architecture of the tool. We use dependant types to represent typed graphs (with implicitly universally quantified variables), assuming *TypeGraph* and *Graph*(i : *TypeGraph*) without defining them. The central part of the application is given by the SST component, which can run a simulation up to a given depth, relying on an interface with a graph transformation tool (GTT) based on incremental pattern-matching, taking a GTS as input (type graph, initial graph and rules), providing functions to get from a graph all the matches (*all*), the new ones (*new*) and the pre-existing ones (*prex*), on a random number generator (RNG), and on a stochastic information source (SIS) interfaced with the SST through a function that represents Δ. We also give a

comparatively concrete characterisation of the data-structures using lists and standard functions for list manipulation — *map* (applying a function to all the elements of a list), *filter* (filtering the elements of a list through a property), *head* (returning the first element of a list) and *append* (returning the concatenation of two lists).

```
typedef Time, Rate, Mean, Deviation = Real
typedef RuleName = String
typedef Match(i:TypeGraph) = (RuleName(i), Graph(i))
typedef Rule(i:TypeGraph)  = (Graph(i), Graph(i))

datatype Distribution = exponential Rate
                      | normal Mean Deviation

interface GTT   % graph transformation tool
     input        tG             : TypeGraph
                  initialGraph   : Graph(tG)
                  rules          : Rule(tG) List
     attributes   #currentGraph  : Graph(tG)

     all    : Graph(i) -> Rule(i) List -> Match(i) List
     new    : Graph(i) -> Rule(i) List -> Match(i) -> Match(i) List
     prex   : Graph(i) -> Rule(i) List -> Match(i) -> Match(i) List
     apply  : Graph(i) -> Rule(i) List -> Match(i) -> Graph(i)

     allMatches    : Match(tG) List
     allMatches    = all #currentGraph rules
     newMatches    : Match(tG) -> Match(tG) List
     newMatches m  = new #currentGraph rules m
     prexMatches   : Match(tG) -> Match(tG) List
     prexMatches m = prex #currentGraph rules m

     initialise    : IO
     initialise =
        #currentGraph = initialGraph
     applyMatch    : Match(tG) -> IO
        #currentGraph m = apply #currentGraph rules m

interface RNG   % random number generator
     randomGenerate : Distribution -> Time

interface SIS   % stochastic information source
     delta  : Match(i) -> Distribution
```

```
component SST       % stochastic simulation tool
     input        maxDepth   : Nat
     attributes   #simTime   : Time
                  #eventList : (Match(GTT.tG)*Time) List
                  #count     : Nat

     evTime : (Match(i)*Time) -> Time
     evTime (m, t)  = t
     evMatch : (Match(i)*Time) -> Match(i)
     evMatch (m, t) = m

     main =
       initialise
       repeat (#count = 1
               while #count < maxDepth
               #count = #count + 1)
          step #count

     initialise : IO
     initialise =
        GTT.initialise
        #simTime   = 0
        evLs       = map mkTimedEvent GTT.allMatches
        #eventList = sortedByTime evLs

     step : Nat -> IO
     step n =
        selEvent   = head #eventList
        selMatch   = evMatch selEvent
        GTT.applyMatch selMatch
        #simTime   = evTime selEvent
        prexMts    = GTT.prexMatches selMatch
        prexEvs    = filter
                     lambda x. member (evMatch x) prexMts
                     #eventList
        newMts     = GTT.newMatches selMatch
        newEvs     = map mkTimedEvent newMts
        #eventList = sortedByTime (append prexEvs newEvs)
        report 'application time' #simTime
        report 'applied rule match' selMatch

     mkTimedEvent : Match(i) -> (Match(i)*Time)
     mkTimedEvent mt =
           distFun = SIS.delta mt
```

```
      randNm  = RNG.randomGenerate distFun
      time    = #simTime + randNm
   return (mt, time)

sortedByTime : (Match(i)*Time) List -> (Match(i)*Time) List
sortedByTime ls =
      e  = head ls
      xs = filter (lambda x. evTime x < evTime e) ls
      ys = filter (lambda x. evTime x >= evTime e) ls
      ws = sortedByTime xs
      zs = sortedByTime ys
   return (append ws zs)
```

6 Application to a simple scenario

We may now consider a very simple scenario (as pictured in Figure 8). In the initial graph, a packet $p1$ is located in the super-node sn waiting to be delivered to the receiver client node $sc2$, whereas a second packet $p2$ is located in the sender client node $sc1$. Eventually packet $p1$ is delivered to $sc2$. Then packet $p2$ can be moved from $sc1$ to the super-node. Finally packet $p2$ in sn is delivered to $sc2$. Figure 8 shows which rules are applied.

We assume a GTS with three rules: *send* (Figure 7(b)), *deliver* (Figure 7(d)), and *clock tick* (Figure 2). We can model the clock behaviour by associating with each *chronos* rule-match a normal distribution δ_i with mean m_i and deviation d_i ($i \in \{sn, sc1, sc2\}$). Rule *send* and *deliver* can be more naturally associated with exponential distributions — hence with exponential rate values, that here we can assume independent of the matches. The application of the algorithm to the initial graph S_0 can start with the computation of $active(S_0) = \{(M_0, dispatch), (sn, tick), (sc1, tick), (sc2, tick)\}$, where M_0 is the subgraph of G_0 associated with the left hand-side of *deliver*. At this stage *send* is not enabled due to the negative applicative condition which is not satisfied ($p1$ is still in sn). When *deliver* wins the time race and is applied, rule *send* becomes enabled. Finally, when *send* has been applied, *deliver* becomes enabled once more, its application leading to the last graph.

7 Conclusion

This paper illustrates the use of generalised stochastic graph transformation systems for modelling VoIP network protocols — with special focus on Skype. A simulation algorithms based on the Event Scheduling Scheme [CL08] has been presented, and its possible implementation using existing tools discussed, relying on a semantic interpretation into generalised semi-Markov processes.

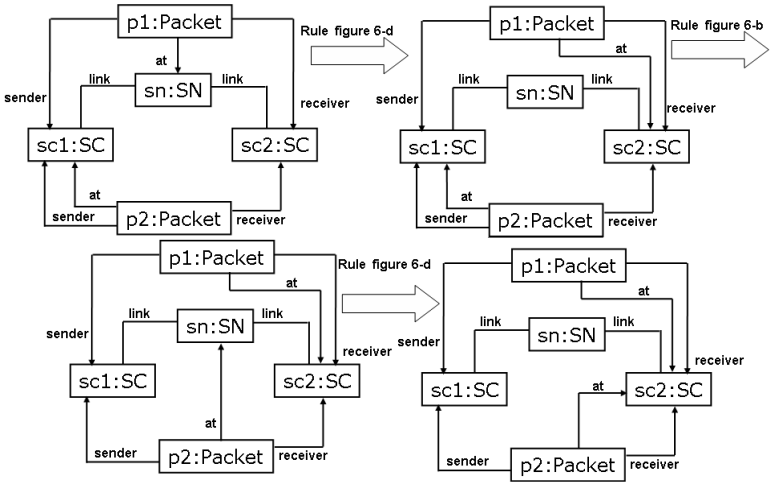


Figure 8: Application scenario

Future work will address the implementation of the simulation algorithm as well as further and extended case studies. A major problem of all stochastic modelling approaches is to find realistic parameters for probability distributions. In order to validate our models we are planning to compare and complement simulations based on stochastic graph transformation with the results of network simulation tools such as NS2 [ISI08] as well as measurements on real systems.

References

[AKK07] M. J. Arif, S. Karunasekera, S. Kulkarni. SOVoIP: True Convergence of Data and Voice Network. *16th ACM WWW conference Banff, Alberta, Canada*, 2007. http://www2007.org/workshops/paper_151.pdf

[BA07] L. G. Birta, G. Arbez. *Modelling and simulation*. Springer, 2007.

[BCM99] P. Baldan, A. Corradini, U. Montanari. Unfolding and event structure semantics for graph grammars. In Thomas (ed.), *Proceedings of FoSSaCS '99*. LNCS 1578, pp. 73–89. Springer, 1999.

[BHRV08] G. Bergmann, A. Horváth, I. Ráth, D. Varró. A benchmark evaluation of incremental pattern matching in graph transformation. In *International Conference on Graph Transformation*. LNCS 5214, pp. 396–410. 2008.

[BS06] S. A. Baset, H. G. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM'06)*. Pp. 1–11. 2006. http://dx.doi.org/10.1109/INFOCOM.2006.312

[CL08] C. G. Cassandras, S. Lafortune. *Introduction to discrete event systems*. Kluwer, 2008.

[DK05] P. R. D'Argenio, J.-P. Katoen. A theory of stochastic systems. Part I: stochastic automata. *Information and computation* 203:1–38, 2005.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of algebraic graph transformation*. Springer, 2006.

[GDJ06] S. Guha, N. Daswani, R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *IPTPS'06: The 5th International Workshop on Peer-to-Peer Systems*. 2006.
http://saikat.guha.cc/pub/iptps06-skype.pdf

[Glo06] Global IP Sound, Inc. Measuring Voice Quality. 2006. White paper.
http://www.gipscorp.com/files/english/white_papers/Measuring%20Voice%20Quality%.doc

[GS04] R. Gupta, A. K. Somani. Pricing strategy for incentivizing selfish nodes to share resources in peer-to-peer (P2P) networks . *Proceedings of the 12th IEEE International Conference on Networks (ICON'04)* 2:624–629, 2004.

[GVH03] S. Gyapay, D. Varró, R. Heckel. Graph transformation with time. *Funamenta Informaticae* 58:1–22, 2003.

[Hec05] R. Heckel. Stochastic Analysis of Graph Transformation Systems: A Case Study in P2P Networks. In *Proc. Intl. Colloquium on Theoretical Aspects of Computing (ICTAC'05)*. LNCS 3722, pp. 53–69. Springer-Verlag, 2005.

[HLM06] R. Heckel, G. Lajios, S. Menge. Stochastic graph transformation systems. *Fundamenta Informaticae* 72:1–22, 2006.
http://www.cs.le.ac.uk/people/rh122/papers/2006/HLM06FI.pdf

[ISI08] ISI, University of Southern California. The network simulator — ns2. 2008. Wikipedia page.
http://www.isi.edu/nsnam/ns

[KL07] P. Kosiuczenko, G. Lajios. Simulation of generalised semi-Markov processes based on graph transformation systems. *Electronic Notes in Theoretical Computer Science* 175:73–86, 2007.

[Lin07] J. Linden. VoIP — Better than PSTN? Technical report, Global IP Sound, Inc, 2007.
http://www.analogzone.com/nett0307.pdf

[LMP04] O. Lysne, J. M. Montanana, T. M. Pinkston. Simple Deadlock-Free Dynamic Network Reconfiguration. *LNCS, SpringerLink* 3296/2005:504–515, 2004.

[MSZ03] S. Merugu, S. Srinivasan, E. Zegura. P-Sim: A Simulator for Peer-to-Peer Networks. *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS03)*, 2003.

[Nel95] R. Nelson. *Probability, Stochastic processes, and queueing theory*. Springer-Verlag, 1995.

[Sky06] Skype Limited. Skype: Guide for network administrators. 2006. http://www.google.co.uk/search?hl=en&q=Skype%3A+Guide+for+network+administrators&btnG=Search&meta=

[Spi07] Spirent Communications, Inc. Measuring Jitter Accurately. 2007. VoIP White paper. http://www.spirent.com

[Tel06] TeliaSonera International Carrier. Ensuring voice quality and integrity in an IP-based network. 2006. VoIP White paper. http://www.teliasoneraic.com/files

[Tur92] R. Turner. *Constructive foundations of functional programming*. McGraw-Hill, 1992.

[VV04] G. Varró, D. Varró. Graph Transformation with Incremental Updates. *Electr. Notes Theor. Comput. Sci.* 109:71–83, 2004.

[XY07] H. Xie, Y. R. Yang. A measurement-based study of the Skype Peer-to-Peer VoIP Performance. In *Proceedings of IPTPS*. 2007. http://reserch.microsfot.com/workshop/IPTPS2007/papers/Xieyang.pdf

[YCSC02] E. Yücesan, C.-H. Chen, J. L. Snowdon, J. M. Charnes. SSJ: a framework for stochastic simulation in Java. In *Proceedings of the 2002 Winter Simulation Conference*. Pp. 234–242. 2002.