Model-Based Stochastic Simulation of P2P VoIP Using Graph Transformation System

Ajab Khan[1], Reiko Heckel[1], Paolo Torrini[1], and István Ráth[2]

[1] Department of Computer Science, University of Leicester
{ak271,reiko,pt95}@mcs.le.ac.uk
[2] Department of Measurement and Information Systems
Budapest University of Technology and Economics
rath@mit.bme.hu

Abstract. P2P systems are characterised by large-scale distribution and high degree of architectural dynamics caused by their lack of central coordination. In such an environment, it is notoriously hard to guarantee a good quality of service. Simulation can help to validate network designs and protocols, but most existing simulation approaches cannot cope with unbounded dynamic change of network topology.

We propose an approach to modelling and simulation of P2P systems based on graph transformations, a visual rule based formalism that has recently been supported by facilities for stochastic modelling and simulation. Focussing on P2P VoIP applications such as Skype, we model alternative solutions to the problem of selection of and connection to super nodes (i.e., the peers acting as servers in the network) and evaluate these through simulation.

1 Introduction

Todays P2P networks [3] present several unique features that differentiate them from traditional distributed systems. Network of hundreds of thousands or even millions of peers are common. They experience a steady flow of peers joining or departing from the network, as well as constant dynamic reconfiguration of network connections.

Large scale, geographically diverse location and peer dynamism present several complex challenges to the network designer. In P2P networks, neither a central authority nor a fixed overlay topology can be used to control the various components. Instead, a dynamically changing overlay topology is used and where control is completely decentralized. Due to the lack of global control and unreliability of the infrastructure, P2P systems are prone to dependability problems. The overlay topology is maintained by cooperation links among nodes. The links are created and deleted based on the requirements of a particular application. Peers are in full control of their local resources and can therefore choose to change or impose new policies regarding their use in the network [1]. A peer may even behave selfishly by not routing traffic for others [2].

In the early stage of the P2P network, most of the applications implemented over the Internet were characterised by the absence of a specific mechanism for enforcing a particular overlay topology [4]. This resulted in the adaptation of inefficient communication schemes such as flooding, or the maintenance of large numbers of connections with

K. Al-Begain, D. Fiems, and W. Knottenbelt (Eds.): ASMTA 2010, LNCS 6148, pp. 204–217, 2010.

other peers. However, it is worth mentioning that situation and approach to P2P overlay topology have significantly changed. Several academic research projects on P2P have realized the importance of selecting, constructing and maintaining appropriate overlay topologies for implementation of efficient and robust P2P systems [5,6,7,4].

Also P2P Voice over IP (VoIP) networks such as Skype [8,9] have started considering more structured topologies by distinguishing client peers from super nodes. This results in a two-level hierarchy: Nodes with powerful CPU, more free memory and greater bandwidth take on server-like responsibilities and provide services to a set of client peers. This approach allows decentralized overlay network to run more efficiently by exploiting heterogeneity and distributing load to machines that can handle the burden. It has also overcome the flaws of the client server model, because of multiple separate points of failure, thus increasing the health of the P2P overlay network.

Building and maintaining a super node based overlay topology is not simple. Rapid architectural chances in both ordinary and super nodes require robust and efficient protocols, capable of self-reconfiguring the overlay topology in spite of both controlled and selfish events like joining, leaving or crashing nodes. In case the P2P is used for VoIP traffic, the network needs to reconfigure fast enough so that Quality of Service (QoS) is not affected [10].

Several questions arise for the design of network protocols: Which super node should a new client peer connect to when joining the network? Can we predict if a super node will be capable of providing VoIP services to all connected nodes? What shall we do when, selfishly, a super node leaves the network? The performance of such a protocol can be measured by answering the question: How many clients are generally provided with good quality connection.

Various solutions have been proposed to these problems, e.g. [11] discussed general design issues however, their focus is on centralized design of such networks, [7] suggested the deployment of super nodes directly managed by content service providers, [4] presented a supper node overlay topology algorithm and validated the approach using the Psim simulator. [2] proposes that an incentive should be given to intermediate nodes and resource owners, [12] proposes to maintain redundant links between peers, [13] propose an autonomous system-aware peer-relay protocol called ASAP, [14] proposes solutions based on changes in routing strategies.

However, peer dynamics and complexity of P2P networks make it difficult and expensive to validate these solutions through testing of real networks or simulation. Geographical distribution of peers, network dynamics and lack of central control make testing difficult and costly. The simulation of network reconfiguration is not easy, as existing simulators do provide very limited support for networks with dynamic topology [12,15].

We propose to model complex network reconfigurations in P2P VoIP networks by means of graph transformation systems and use a new approach to the stochastic implantation of such systems to evaluate the performance of network protocols. 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 naturally be medalled by graph transformation, in a visual and rule-based formalism [10,12]. Stochastic simulation techniques for validation have been developed in [10].

In this paper we are going to present a case study based on the popular VoIP application Skype and discuss how to face some of the challenges posed by it.

2 Case Study: Skype Network

Skype is a P2P VoIP network developed by KaZa in 2003. It has currently more than 170 million registered users, 10% of which are usually online. Skype allows registered users to make voice calls and send messages, files or video to other users. It has the ability to encrypt the calls and store the user information in decentralized form [18]. Skype is a proprietary P2P protocol which competes against open protocols such as SIP and H.323. Features such as the ability to overcome the problem of the network address translation (NAT) and firewalls make Skype very attractive. It also allows users to call switch telephone network (PSTN) numbers at much lower cost. The main difference between Skype and other VoIP applications is that it operates on the P2P model rather than the traditional client server model. The Skype directory structures are completely decentralized which enable the system to scale easily to large numbers of users without requiring complex infrastructure [4].

The first detailed study of the Skype network architecture was performed in 2004 [18]. After this several new version were released, but the core network features remain the same. Skype network nodes are distinguished into Skype Clients and Super Nodes. The network nodes supporting Skype peers are divers in their computational power, storage capabilities, and most importantly the network connection type and bandwidth. Peers supplied with sufficient resources can be promoted to the role of Super Node while continuing to function as Clients. Super nodes form an overly network amongst themselves, whereas each client has to register with a Registration Server and select one of the super nodes as their server. The client will use their chosen super node as a contact to receive or issue calls or, if hidden behind a firewall, even as router for the actual VoPI traffic. The Registration Server is the only central server in the network, responsible for storing user names and passwords, authenticating users on login, and providing them with the addresses of super nodes to make their connection with the network. All information about user's online status is stored in a distributed way by the super nodes in the network, which improves scalability and stability even if information can be sometimes out of date.

The population of super nodes in the network is not determined by demand but based on the availability of bandwidth and their reachability [8]. A network may have more super nodes than strictly necessary if these resources are plentiful. Due to the proprietary nature of Skype, little information is available about codecs but the analysis in [18] claims that Skype uses 5kbps to 16kps bandwidth whereas [19] states that bandwidth consumed is 25kbps whenever a VoIP call is in progress. The clients also send keep-alive messages to the super node and receive back replies in order to check whether the super node still exists. In case the super node has left the network, the client has to reconfigure and try another super node for establishing a connection. The super node, based on the available free bandwidth, may allow or refuse new connections.

Both client and super node can leave the network either by shutting down the computer (crashing) or by using the proper exit procedure available in the application's user interface.

3 A Graph Based Model for Skype

We use graph transformations to model the structural evolution of the Skype network. As one of the most basic models for entities and relations, graphs are a representations of structural models. Formally, a graph consists of a set of vertices V and a set of edges E such that each edge $e \in E$ has source and target vertex $s(e)$ and $t(e)$ in V , respectively. More advanced notions allow for nodes and edges to be attributed with textual, Boolean or numeric data [16]. Graphs occur at two levels: type level and instance level. A type-level graph is comparable to a class or ER diagram containing the types of nodes and edges, declarations of attributes, etc. Instance graphs represent the states of the system, typed over the type graph. With graphs as states, transformation rules provide state changing operations [12,10].

The type graph *TG* in Fig 1 represents a model of the architecture of Skype as described earlier. It defines the types for registration server (*RS*), super node (*SN*), Skype client (*SC*), and their common supertype. The node type *LK* is used to model links between *SN* and *SC* while *OV* represents overlay connections between existing *SN*s. The edges for *registration* and *RS-overlay* are used to show the connection of the *SC* and *SN* with *RS*.

In the model, whenever a new *SC* joins the network, first it has to get registered with the *RS* and in the next step it has to select one of the *SN* as local host. The local host will be used for querying the network and to transfer the actual payload of the voice packet. In the model *SC*s with bandwidth more than the *1.5Mbps* are promoted to the new role of the *SN*. The model does not restrict the number of the *SN* in the network.

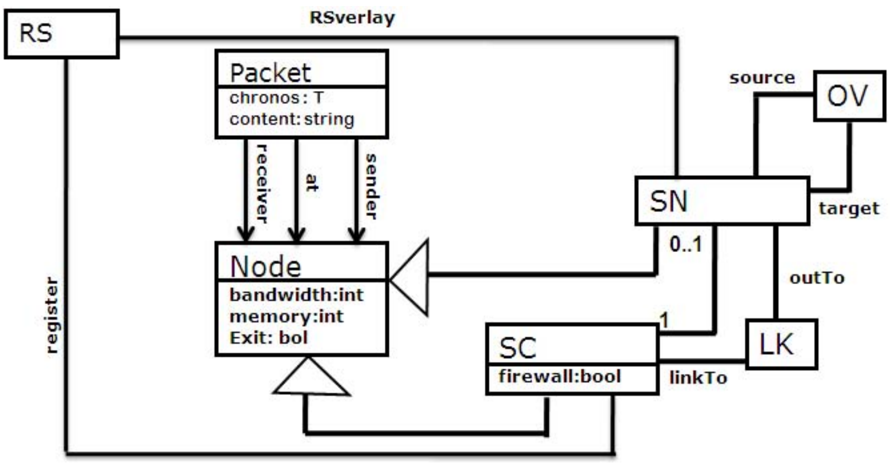


Fig. 1. Type graph

Based on this architecture we model two different approaches to connect an *SC* with an *SN*. In the first approach, we randomly select any *SN* and if it has the capacity to accept a new connection, (depending on the available bandwidth), a link is established using *LK* between the *SC* and *SN*. In the second approach, we establish a link between *SC* and *SN* based on the latency in communication between the *SC* and *SN*. We measure the latency by *Packet* carrying a time stamp. If the round-trip time taken by the packet is less than 300*ms* and the bandwidth of the *SN* permits a new connection, the link *LK* is established between the *SC* and *SN*.

In order to model VoIP traffic we assume a *codec* using *60kbps* of the bandwidth of the *SN*, such that all the VoIP traffic is routed through the *SN*. We randomly increase and decrease the bandwidth of the *SN* in order to model the running VoIP traffic. If an *SN* departs from the network either by crashing or controlled exit, the model is capable to reconfigure the *SC* and link it back to a new *SN* based on one of the two approaches discussed above.

The objective of modelling these two protocols of connection is to evaluate and compare their performance in terms of the number of *SC*s enjoying a connection with sufficient bandwidth. The model will also provide information regarding the overall number of *SN*s and *SC*s in the network.

4 P2P Network Connection as Graph Transformation

A graph transformation rule $p : L \longrightarrow R$ consists of a pair of TG-typed instance graphs L, R such that the intersection $L \cap R$ is well defined. The left-hand side L represents the pre-conditions of the rule whereas the right-hand side R describes the post-conditions. Their intersection represents the elements that are required, but not destroyed, by the transformation [12]. Graph transformation rules also use negative application conditions *(NACs)*. A NAC assures that the rule will only be applied if the pattern specified NAC does not match the graph [12,10].

We are now going to introduce a set of transformation rules based on a simple network connection scenario. Here, due to limitation of space we are not introducing the rules for promotion of *SC* to *SN*, crashing, and controlled exits. However, in the simulation all these rules are provided in order to give results on the complete model.

Rule in Fig 2 create, Skype nodes. This rule creates new Skype nodes and assigns randomly a bandwidth between *56kbps* and *2Mbps*. Nodes with bandwidth equal or higher than than 1.5 Mbps are promoted to the role of *SN*.

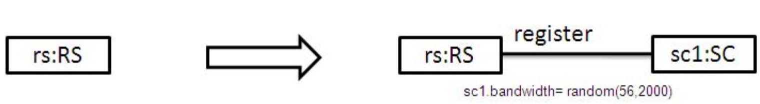
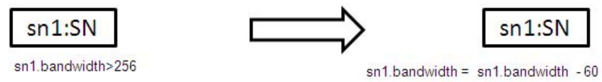
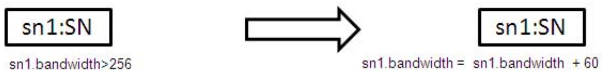


Fig. 2. Create Skype nodes

Rule in Fig 3: create, remove VoIP traffic in overlay network. Rule (a) creates new traffic worth 60kbps at the *SN*. This is an average value for ITU-T codecs, each of which has its own data rate [21]. This means that whenever rule (a) is executed, it reduce the bandwidth of the *SN* by 60kbps. Since the *SN* to which this rule is applied will be selected at random, it will create the effect of random traffic in the overlay network. Rule (b) increases the bandwidth by adding the 60kbps, corresponding to a decrease in VoIP traffic load on the *SN*.



(a) Create traffic of *60kbps*



(b) Remove traffic of *60kbps*

Fig. 3. VoIP traffic in the SN overlay network

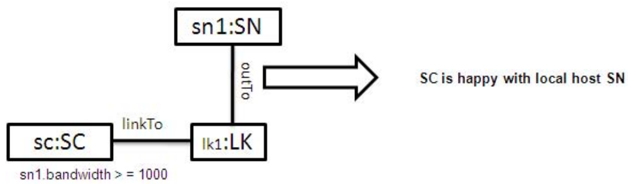
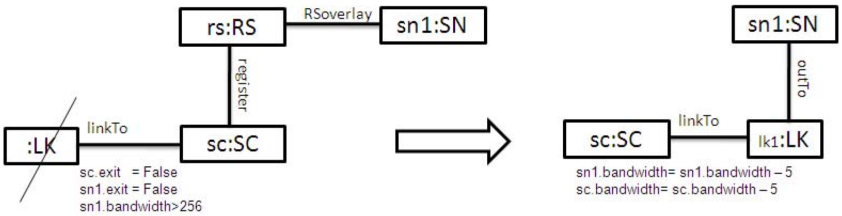


Fig. 4. Probe rule to find *happy SC*

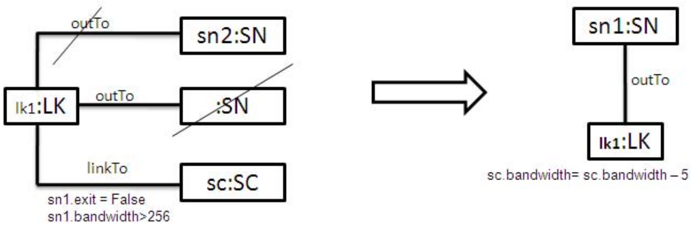
Rule in Fig 4: find "happy" Rule is used as a probe to find those *SC* clients currently connected to an *SN* with bandwidth more than 1Mbps. This is necessary as this will make sure that the local host is in a position to accept new VoIP calls.

Rules in Fig 5: connect SC with SN, reconfigure with new SN. Rule (a) connects *SC* to the randomly chosen *SN* provided that the latter is not currently in the process of leaving the network. To check this we use a Boolean attribute *exit*. If this attribute is true then the *SN* will not accept new connections. The rule also checks the bandwidth of the *SN* and allows connection only if it has a more than *256kbps*. The rule cannot be applied to already connected *SC* due to the negative application condition shown by a crossed out node *LK*.

Rule (b) reconnects an *SC* to a new *SN* if the *SC* was disconnected due to either selfish exit of the *SN* or as a result of local load management. This rule use two NACs, the first to make sure that the *LK* node has lost its connection to the *SN* and the second to ensures that the new, randomly chosen *SN* does not have any connection with the *LK*. This rule also checks that the bandwidth of the selected *SN* is more than the minimum *256kbps* and it is not in the process of controlled exit. If all these condition are satisfied then the SC can be connected to the new *SN*.



(a) Connect SC to SN



(b) Recofigure to connect to new SN

Fig. 5. SC connection to SN based on random approch

Rules in Fig 6: create, send, return time stamped packet, connect with SN and reconfigure with new SN. Rule (a) creates a packet $p1$ and sets the time stamp (*chronos*) attribute of *p1* and *SC* to the system time. The packet *p1* is transmitted to a randomly selected *SN*.

Rule (b) returns the packet with contents *AcK* if the current bandwidth of the *SN* is more than the minimum required and it is currently not in the process of controlled exit.

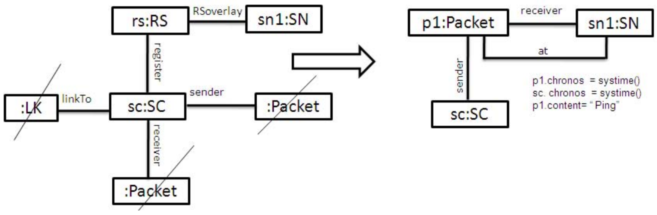
Rule (c) connects the *SC* to the *SN* if the packet received has content *AcK* and the difference between the time stamps at the packet and the current time is no more than 300ms as per the ITU-T VoIP requirements. This packet is used to find the round trip delay between the *SC* and *SN*. As standard connection cost, the bandwidth of the *SN* and *SC* is reduced by 5kbps. This helps the *SC* to select an *SN* based on the latency along with other parameter such as bandwidth and exit.

Rule (d) rejects the selected *SN* if the latency is higher than the acceptable 300ms.

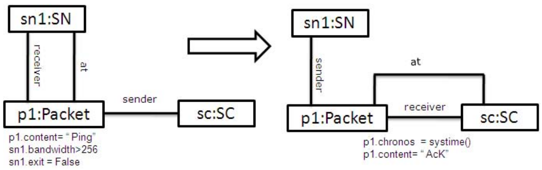
Rule (e) returns the packet with content *DnY* if either the bandwidth is less than required or the *SN* is in controlled exit. Rule (f) deletes the corresponding packet. In this case the procedure starts again from rule (a).

Rule (g) reconnects an *SC* to a new *SN* if the *SC* was disconnected due to departure of SN (Selfish exit or laod managment). This rule use four NACs, the first to make sure that the *LK* node has lost its connection to the *SN* and the second to ensures that the new, randomly chosen *SN* does not have any connection with the *LK*, third make sure that no request is sent, the last ensures that SC is not waiting for request reply from SN. This rule also checks that the bandwidth of the selected *SN* is more than the minimum *256kbps* and it is not in the process of controlled exit. If all these condition are satisfied then the SC can send a request packet to the new *SN*.

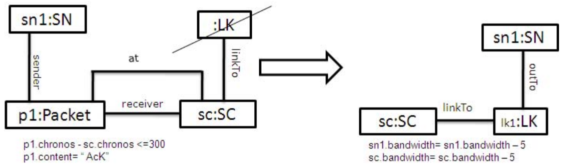
Based on the above transformation rules We consider a simple scenario (as pictured in Fig 7)in order to show the applicability of the rules. In the initial graph, the super nodes *sn1* and *sn2* are registered with the registration server. As the first rule is applied, a



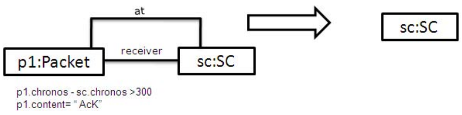
(a) Create and Send time stamped *ping* packet



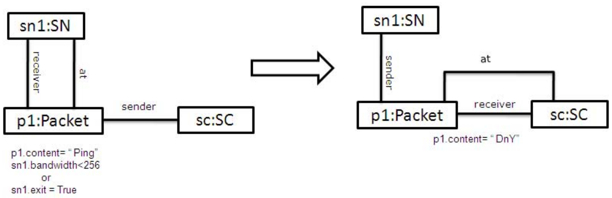
(b) Return *reply* packet with *AcK*



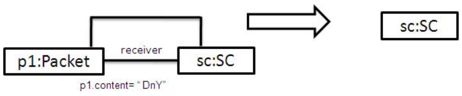
(c) Connect SC with the SN



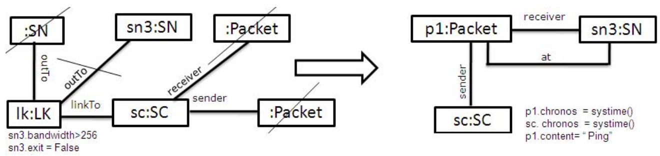
(d) Reject SN based on *latency*



(e) Return *reply* packet with *DnY*



(f) Delete *reply packet* try new SN



(g) Reconfigure to connect *new SN*

Fig. 6. SC connection to SN based on latency

new Skype client *sc* joins the network by registering with the server *RS*. In the following step, the client *sc* has to select one of the existing super nodes. In this example we show the random approach. As the rule Fig 5(a) is applied, the client *sc* gets linked with *sn1*. With an execution of the *uncontrolled departure* rule, *sn1* leaves the network and the client *sc* remains disconnected. As the reconfiguration rule Fig 5(b) is applied, the client *sc* gets reconnected, this time with super node *sn2*. Finally, the last transformation shows that when the traffic simulation rule Fig 3(a) is applied the bandwidth of the super node is reduced.

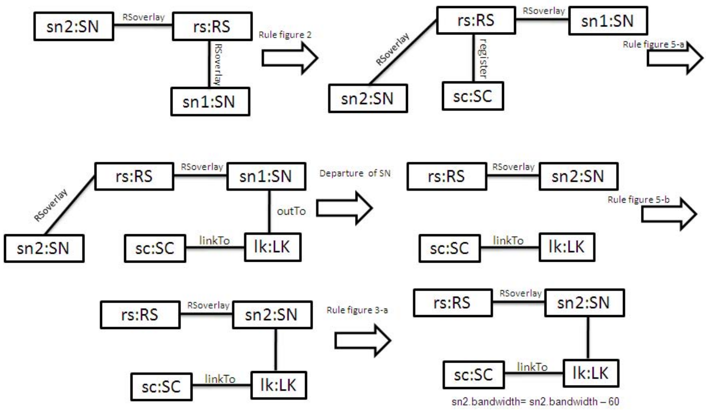


Fig. 7. Application scenario

5 Stochastic Simulation of Graph Transformation System

The traditional approach in network simulation is to model the network in terms of nodes and links, where each link is individually associated with bandwidth and delay properties. When this approach is used to simulate large P2P networks, the number of events to be processed can easily lead to problems, particularly in relationship with topological reconfiguration due to peer dynamism.

Stochastic graph transformation [12] can make it easier to model architectural reconfiguration and non-functional properties such as performance and reliability. A stochastic graph transformation system (SGTS) is a graph transformation system (GTS) where each rule is associated with a positive real number representing the rate of the exponentially distributed delay of its application. Graph transformation can not only model these networks but it also support a number of validation and verification techniques. Model checking based on CSL and stochastic simulation based on translation of to Markov chains were introduced in [17] for SGTS. Model checking is useful to formally verify the abstract properties of processes, but this can be hard in case of complex examples. On the other hand, Monte-Carlo stochastic simulation is typically based on the execution of particular processes, which are selected probabilistically by means of a random number generator (RNG).

Let us consider that a $\mathcal{S}_{\mathcal{G}} = \langle \mathcal{G}, \mathcal{F} \rangle$ is a *generalised stochastic graph transformation system* whenever $\mathcal{G}$ is a GTS and $F : E_{\mathcal{G}} \rightarrow (\mathbb{R} \rightarrow [0, 1])$ is a function which associates with every event in $\mathcal{G}$ a general cumulative probability distribution function. We assume that $F(e)(0) = 0$ (*null delay* condition) [10]. Moreover, the probability distribution is dependent on the event (*rulename and match*) rather than just the rule. The concept of the *SGTS* is explained [17] in detail. Our interest in stochastic graph transformation systems is closely associated with their simulation, where the stochastic aspect is useful in order to resolve the non-deterministic character of ordinary GTSs.

We simulate our model using GraSS (for Graph-based Stochastic Simulation), a new tool introduced in [20]. The tool has been developed in Java-Eclipse, as plugin of a graph transformation engine called VIATRA. VIATRA [22] relies on a RETE-style implementation of incremental pattern-matching, in which precomputed matching information is stored and updated as transformation proceeds. The architecture of the tool is shown in Fig 8. Essentially, the stochastic engine receives the set of enabled rule matches (i.e. the active events) from the transformation engine, turns them into timed events, by assigning to each of them an expected time value, randomly determined on the basis of the probability distribution which is associated with the event type, and sends the events that has been scheduled first back to the transformation engine for execution.

In GraSS a GTS is represented as a VIATRA model, consisting of the model space with the current graph and the transformation rules. Moreover, GraSS takes as input an XML file with the definitions of the distributions associated with transformation rules and events, as well as the list of the rules with empty post-conditions that are to be used as probes. Additional parameters needed for a simulation run are provided to GraSS as part of the VIATRA model (see [20]).

In this experiment we use only exponential distributions, and therefore we only need to associate each transformation rule with a rate. We run several simulations of each of the two approaches, varying the rate of the rule in Fig 2 by a factor of {10, 100, 200,

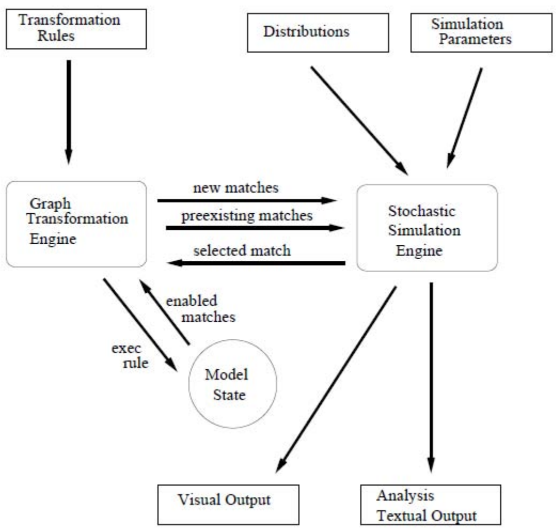


Fig. 8. GraSS

```
gtrule Rule17_SC_Happy_with_SN_Bandwidth() =
  {
        precondition pattern lhs(SC,LK,SN,BW) =
        {
              SN(SN);
              SC(SC);
              LK(LK);
              find LK_LinkTo(LK,SC);
              find LK_OutTo(LK,SN);
               find SN_Bandwidth(SN,BW);
               check((toInteger(value(BW)))>1000);
        }

        action {
                    println("SC is happy with existing SN");
                          }
  }
```

Fig. 9. Probe rules VIATRA code

400, 1000, 5000, 10000}. The rules in Fig. 3(a) are used with fixed rates of 400 and those in Fig. 3(b) with rates of 200 in both versions of the model. The rates has been doubled in order to explore the effect of increased traffic in the network. The rules in Fig 5 and Fig 6 have been used with rates of 200 respectively. All the other rules, such as *uncontrolled exit* and *controlled exit*, *load management*, and *downgrade* (not presented in this paper due to space limitation) are kept at a rate of 1.

In order to collect the statistics of the simulation, rules with empty postconditions are used as *probes*. Each probe rule returns the number of its matches in the current graph for each state of the transformation system. The probe rules used in this paper are pictured in Fig 4, whereas their VIATRA code can be seen in Fig 9. The textual output of a simulation experiment consists of SSJ *TallyStore* class reports [23].

GraSS can be used to run batches of independent simulations, obtained by restarting the initial graph for a given number of times. The maximum depth of the simulation runs in the batch can be given either in terms of simulation time or of the number of transformation steps. While running individual simulations, GraSS computes statistics of the probes, by collecting average, maximum, minimum and standard deviation values for each of them. Over each batch of runs, GraSS computes average, standard deviation and a confidence interval associated with each variable. GraSS can also be used to automatically generate a sequence of independent simulation batches, each with different distributions associated to sensistive rules. It then provides a final report, over the batches, with a confidence interval for each probe, on the average value of that probe in a batch. Numbers of runs for batch, maximum depth and sensitive rule variations are simulation parameters that, together with the graph transformation system and the probes, define a simulation experiment.

In this experiment we compared two models, based on different approaches for connecting clients with supernodes, along the line we discussed earlier. Each model has been tested by running batches of simulations, varying the rate of the node creation rule (Fig 2). Each batch consists of 6 independent runs, each bounded by a time limit of 0.1 second

We programamed GRASS to automatically generate independent batches of simulation for each model, with node creation rates ranging over $x \in \{1, 10, 100, 1000\}$. We produce confidence intervals based on t - distributions, with a confidence level of 95%.

6 Simulation Results

In this experiment we compared two approaches for connecting clients to super nodes. Each approach is presented by a model that was tested through 4 variations of the rate x for the node creation rule (Fig 2), ranging over $\{1, 10, 100, 1000\}$. For each variation we performed twelve runs with a time limit of 0.1 seconds of simulated time each. The results are reflected in the two tables 1 and 2. The 1st column shows the rate of the creation rule. The 2nd column shows the lower limit. The 3rd column shows the average number of the *SC* nodes in the network. The 4th column shows the upper limit of the confidence interval. The 6th column shows the average total number of *SN* nodes in the network. The 8th column shows the percentage of the linked SCs with respect to total number of SCs in network. The 10th column shows the average number of clients who are currently happy with their existing SN.

We compare the performance of both approaches with respect to four measurements: the total number of SC nodes in the network, the number of super nodes in the network, the percentage of linked SC nodes, and the ratio of happy peers.

We have used t-distribution for computing the 95% confidence interval because the size of the data is small as we had 6 run for each of the rate. The computed confidence intervals are reflected in the tables below showing the respective lower and upper limit for each of the measurement.

The simulation results show for both models a remarking degree of scalability, but when node creation is more rapid, the latency-based model ends up with a higher proportion of SC nodes which are not linked to super nodes (yet). This results in decreasing the proportion of happy clients. This effect is very pronounced at a node creation rate of 1000, where the total number of connected SC nodes actually drops when the network is flooded with ping messages by new SC nodes looking for a good SN to connect to. Thus, the randomised approach performs better in terms of registering and promoting clients. This fits the intuition as the latency-based approach involves a more complex linking process while not harvesting the benefits.

Table 1. Random Model

Rate	Lower Limit	Avg.SC	Upper limit	Lower Limit	Avg.SN	Upper Limit	Lower Limit	%Linked	Upper Limit	Lower Limit	Avg.Happy	Upper Limit
1	0.218	0.860	1.502	1.000	1.000	1.000	0.610	0.706	0.803	0.165	0.166	0.382
10	4.465	5.097	6.155	1.235	1.646	2.403	0.709	0.789	0.887	0.532	2.746	37.930
100	40.761	48.185	55.609	3.493	5.112	6.731	0.883	0.910	0.938	18.700	28.316	37.930
1000	452.759	474.009	495.259	17.076	19.324	21.571	0.951	0.958	0.965	118.200	164.407	210.600

Table 2. Latency-based Model

Rate	Lower Limit	Avg.SC	Upper limit	Lower Limit	Avg.SN	Upper Limit	Lower Limit	%Linked	Upper Limit	Lower Limit	Avg.Happy	Upper Limit
1	0.687	0.814	0940	0.896	1.065	1.235	0.511	0.716	0.9221	0.096	0.166	0.441
10	2.761	5.506	8.252	0.832	1.479	2.126	0.682	0.746	0.810	0.158	2.150	32.157
100	45.922	51.660	57.398	4.818	6.419	8.019	0.865	0.883	0.901	22.620	27.389	32.157
1000	462.799	484.934	507.069	72.127	101.4608	130.794	0.168	0.459	0.749	43.613	98.797	153.980

7 Conclusion

In this paper we have outlined our simulation approach based on stochastic graph transformation. We have applied it to modelling and simulating some aspects of P2P VoIP network protocols, and we have performed our experiments with the GraSS/VIATRA tool [20]. We have compared two configuring approaches. The more sophisticated one does not seem to perform better as compared to the random based one, the reason being that the model does not include geographic information about where clients and supernodes are located. In reality, whether a client is linked to a nearby supernode, has definitely significant effect on the quality of service.

As future work, we are planning to extend the model and include spatial information in order to provide latency results not only based on network traffic but also on locations with respect to network topology. We are also planning to extend the model in order to include notions of jitter, packet loss and echo, along the lines of [10], and to compare a number of different design solutions to problems such as promotion of clients to super nodes, routing, load balancing, selfish exit, and cooperative exit from the network, in order to investigate their tradeoffs and benefits.

References

1. Li, C.J.: Computation in Peer-to-Peer Networks. Department of Computer Scince. University of Saskatchewan, Canada
2. Gupta, R., Somani, A.K.: Pricing Strategy for Incentivizing Selfish Nodes to Share Resources in Peer-to-Peer (P2P) Networks. In: Proceedings of the 12th, ICON 2004 (2004)
3. Milojicic, D.S., et al.: Peer-to-Peer Computing. Technical Report HPL-2002-57, HP Labs, Palo Alto (2002)
4. Montresor, A.: A rubust Protocol for Building Superpeer Overlay Tolologies. Department of Computer Science, University of Bologna, Italy, UBLCS-2005-8 (2004)
5. Dabek, F., et al.: Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. In: Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS), Schloss Elmau, Germany, May 2001. IEEE Computer Society, Los Alamitos (2001)
6. Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location and Routing for Large-ScalePeer-to-Peer Systems. In: Proc. of the 18th Int. Conf. on Distributed Systems Platforms, Heidelberg, Germany (November 2001)
7. Zhao, B., et al.: Tapestry: A Resilient Global-scale Overlay for Service Deployment. IEEE Journal on Selected Areas in Communications (2003) (to appear)
8. Guha, S., Daswani, N., Jain, R.: An Experimental Study of the Skype Peer-to-Peer VoIP System. In: IPTPS 2006: The 5th InternationalWorkshop on Peer-to-Peer Systems (2006), `http://saikat.guha.cc/pub/iptps06-skype.pdf`
9. Skype limited. Skype: Guide for Network Administrators (2006)
10. Khan, A., Torrini, P., Heckel, R.: Model-based Simulation of VoIP Netowrk Reconfiguration using Graph Transformation System. In: EASST, ICGT, vol. 17 (2009)
11. Yang, B., Garcia-Molina, H.: Designing a Super-peer Network. In: Proc. of the 19th Int. Conf. on DataEngineering (ICDE), Bangalore, India (March 2003)
12. Heckel, R.: Stochastic Analysis of Graph Transformation Systems: A Case Study in P2P Networks. In: Van Hung, D., Wirsing, M. (eds.) ICTAC 2005. LNCS, vol. 3722, pp. 53–69. Springer, Heidelberg (2005)
13. Ren, S., Guo, L., Zhang, X.: ASAP: an AS-Aware Peer-relay protocol for high quality VoIP. In: Proc. of the 26th Int. Conf. on Distributed Computing Systems (ICDCS 2006), Lisbon, Portugal, July 4-7 (2006)
14. Lysne, O., Montanana, J.M., Pinkston, T.M.: Simple Deadlock-Free Dynamic Network Reconfiguration, LNCS. Springer, Heidelberg (2004), SpringerLink 3296/2005:504515

15. ISI, University of Southern California. The Network Simulator-NS2 (2008), Wikipedia Page http://www.isi.edu/nsnam/ns/
16. de Lara, J., et al.: Attributed Graph Transformation with Node Type Inheritance. Theor. Comput. Sci. In Fundamental Aspects of Software Engineering 376(3), 139–163 (2007)
17. Heckel, R., Lajios, G., Menge, S.: Stochastic graph transformation systems. Fundamenta Informaticae 72, 1–22 (2006), http://www.cs.le.ac.uk/people/rh122/papers/2006/HLM06FI.pdf
18. Baset, S.A., Schulzrine, H.G.: An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In: Proceedings of the 25th IEEE International Conference on Computer Communications, INFOCOM 2006 (2006), http://dx.doi.org/10.1109/INFOCOM.2006.312
19. Idrees, F., Khan, U.A.: A Generic Technique for Voice over Internet Protocol (VoIP) Traffic Detection. IJCSNS International Journal of Computer Science and Network Security 8(2) (February 2008), http://paper.ijcsns.org/07_book/200802/20080207.pdf
20. Torrini, P., Heckel, R., Rath, I.: Stochastic Simulation of Graph Transformation Systems. In: Proceeding of International confrence of Fundamental Approaches to Software Engineering FACE 2010 (accepted 2010)
21. Ahson, S.A., Ilyas, M.: VoIP Handbook, Application, Technologies, Relibality and Security. CRC Press, Boca Raton (2009)
22. Bergmann, G., Őkrős, A., Ráth, I., Varró, G.: Incremental pattern matching in the VIATRA model transformation system. In: GraMoT 2008 (2008)
23. L'Ecuyer, P.L., Meliani, L., Vaucher, J.: SSJ: a framework for stochastic simulation in Java. In: Proceedings of the 2002 Winter Simulation Conference (2002)