Modeling User Behavior: A Layered Approach*

Helmut Hlavacs and Gabriele Kotsis
Institute of Applied Computer Science
and Information Systems
University of Vienna
[hlavacs|gabi]@ani.univie.ac.at

Abstract

The simulation of computer systems requires representative, reliable workload models. When simulating computer systems or components that are influenced by user behavior, this very behavior has to be modeled by using mathematical means. In this article, we propose a user behavior model framework that is constructed in a top down manner, consisting of various layers. Layers offer services to the next higher layer and require services from the layers below. The framework is meant to enable the modeler to plug in his own models at the layers of his choice, thus choosing the right balance between the simulation complexity and creating representative results. We give a description of the layered framework and the corresponding methodology. The approach then will be demonstrated on modeling HTTP traffic to be used in network traffic simulation.

1 Introduction

Future computer systems and network components will face a mixture of different workload types, varying from simple to heavy. Especially the upcoming digitalization of multimedia will impose tight constraints on the quality of service (QoS) that the computing and network resources have to provide. If those requirements are not met, this will cause user dissatisfaction and may well lead to the loss of customers or may render a certain computer system worthless.

The goal is to apply capacity management and performance tuning techniques to balance computer and network components in such a way that important constraints are met while keeping the costs at a minimum. Simulation is a means of evaluating the effects of those management activities, which requires reliable, prototypical models for the generation of artificial workloads. Such workload models can be defined in various ways, from simple, static models to sophisticated dynamic or generative models. Low level models often try to mimic the observed workload as a stochastic process with similar first- and second-order statistics. Models for interactive systems may also take into account the behavior of users, who start, manipulate and stop applications, causing workload in lower-level computer and network components. When dealing with various types of models, it may be desirable to store them in libraries and to retrieve them at simulation start in a convenient manner.

In this paper, a framework for user behavior models for interactive computer systems is presented. They are meant for the simulation of network traffic, stand-alone computer systems and any mixture of these types.

The remainder of the paper is organized as follows. In Section 2, we give a summary of related work in the area of user behavior oriented workload modeling. In Section 3, we describe the targeted simulation architecture and the different layers of the framework. Section 4 sketches the corresponding modeling methodology. In Section 5, we give an example on how to apply the approach in a case study on modeling HTTP traffic. We conclude with an outlook on future work.

2 Related Work

Models for generating realistic computer and network workload have been under study for many years. Many models are represented by stochastic processes (Jagerman, Melamed, Willinger [10], Hlavacs, Kotsis, Steinkellner [9]) only, yielding the time points, where workload arrives at low level computer components such as the Ethernet physical layer or a WWW server, as well as the size of the arriving workload, e.g. the amount of traffic to be transfered over the network.

User behavior models try to catch the sequence of user interactions at a higher level. Such models will in gen-

*This work was funded by the ESPRIT IV research program of the EC under grant EP28425 (BISANTE).

eral be hierarchical, as the workload generated at a higher level will result in a stream of workload requests at a lower level. The PACFGs (Raghavan, Joseph [14]) are an example for such a hierarchical workload description. In Calzarossa et. al. [3], a layered framework for the modeling of workload for parallel systems has been defined. Another popular method for catching inter-command dependencies is to apply Markov chains for user behavior modeling (Menasce, Almeida, Fonseca, Mendes [11], Calzarossa, Marie, Trivedi [5], Calzarossa, Haring, Serazzi [4], Chen [7]). A more static approach has been taken in Noethe [13], where each user session at the observed computer system is represented by an n-dimensional vector, each vector component containing the percentage of commands of a certain type. A similar approach is taken in Yan et. al. [16], where each session at the observed web server in assigned an n-dimensional vector, each component denoting the number of times, a certain web documented has been requested within the session. In Crovella [8], users are represented by so-called user equivalents, simple two-level heavy tailed On-Off processes. In Catledge, Pitkow [6], each MOSAIC session is represented by the slope of a regression line, obtained by plotting the average path length per site against its frequency.

3 Modeling Framework

The major drawback of the models found in the literature is the fact that only single model types are used. Whereas one particular model type might cover a certain field of application sufficiently, it might be insufficient for other applications.

The goal of the modeling framework is to find sets of layered models that can be plugged onto each other to represent some typical user behavior that has been either observed, or that is projected to be seen in the future. The models will be created top down, starting always at the highest layer and then going down to the chosen level of detail, each layer adding another level of detail to the model. Figure 1 shows the framework layers.

What to do next is passed from the highest layer down to the lowest by triggering actions. If such an action is triggered for layer i by the next higher layer $i+1$, layer i starts triggering a stream of actions for the next lower layer $i-1$ and waits for results from it. Results are then processed and, if necessary, passed on to layer $i+1$. The lowest layer in this stack generates the workload for the resource under observation. If the modeler chooses to skip intermediate layers, e.g. because the system under study does not require a model at that level of detail, those layers will be substituted by dummy layers, which simply pass on the actions.

The more layer models are plugged-in, the more detailed will be the workload description and the more accurate will

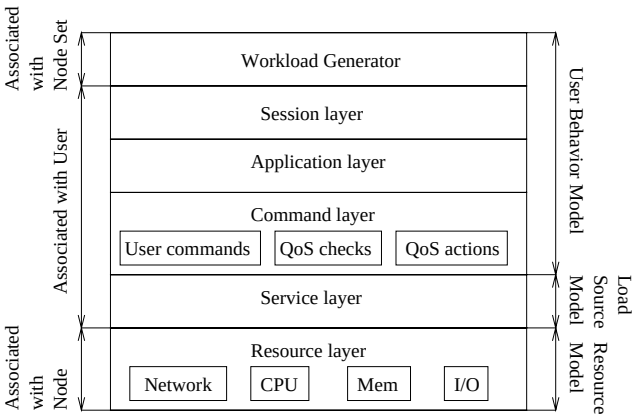

Figure 1. Framework layers.

be the result, leading also to more events to be simulated, increasing the simulator complexity, both w.r.t. CPU time consumed as well as memory requirements.

3.1 Simulation Architecture

The simulation environment is assumed to work in an event-driven fashion. Events denote a piece of atomic work that is scheduled to happen at a specified virtual time point t_i. Generating events of any kind is modeled by the interarrival time of such events and the type of the event itself. At each layer, thus a stream of events is produced.

The simulation architecture is assumed to consist of the simulation kernel, containing the functionality to schedule and execute events, the user models, defining the layers, and terminal equipment or nodes. Within the framework, layer models of a certain type will be treated like model classes. At run-time, instances of these classes will be created and linked together.

At simulation start, the workload generators will create user sessions and will place them to one of the available nodes. The user sessions will then start applications and thus create workload, which will be passed on to the resource layer model associated with the node.

3.2 Terminal Equipment and Node Sets

Terminal equipment or nodes denote the hardware that applications run on. They are also members of the network topology under observation. Terminal equipment is either created and connected to the network topology before simulation start, or is defined as a template. Template instances are then created at run time and are dynamically connected to the network topology at some predefined nodes (dial-in nodes). Terminal equipment can also move around and

might be passed from one dial-in node to the other (handover). Each terminal equipment must be associated with a resource layer model.

Node sets consist of one or more terminal equipments. Static node sets consist of pre-defined nodes that are instantiated at simulation start. Dynamic node sets consist of node templates that are instantiated at run-time as the simulation evolves. Each workload generator must be associated with either one node or with one node set, choosing one of the nodes to host the next user session at run time.

3.3 Layer Signals

The idea behind this framework is to be able to construct libraries holding models of any type. For example, a Markov chain could be used to model the application layer, while the command layer could be modeled by a Poisson process. When actually performing simulation, the user should be able to choose any model for the various layers without being forced to know about the model details.

The different layers though must be able to communicate with each other. From an abstract point of view, this can be achieved by sending signals from one layer to the other. A set of signals thus must be defined for each layer that is used for communication. Each layer model then must be able to deal with such a set of signals, either by doing nothing, sending a signal to another layer or by scheduling one or more events.

At the beginning, a model instance has to be created (initialization), but is still inactive and must wait for a start-signal. Once it has received this signal, a model instance will become active in sending signals and/or generating events for simulation. Such an activity can be stopped with the possibility to become active again, and finally the model instance can be destroyed.

The corresponding standard signals causing the change of state of a model instance and notifying other instances of such state, that have to be interpreted by all layers are:

- SIG_INIT: tells the next lower layer model instance to initialize itself.
- SIG_START: tells the next lower layer model instance to start its activities.
- SIG_STOP: tells the next lower layer model instance to stop its activities.
- SIG_STOPPED: tells the next higher layer model instance that the sender has stopped its activities.
- SIG_END: tells the next lower layer model instance to terminate/destroy itself.

In addition, the service and resource layers must be able to handle additional signals (SIG_REQ, SIG_SEND, SIG_RECV) which indicate the actual data transfer.

Signals can be transported from instance to instance by various means, depending on the programming language used for implementation and the underlying simulation kernel. An object-oriented implementation might choose member functions that are called either directly by other instances or by a central scheduler, as, for example, is done within ns (Bajaj et. al. [2]). Other simulator kernels will allow messages that are passed from one entity to the other, like, for example, PARSEC (Bagrodia et. al. [1]), OPNET ([12]) and OMNeT++ (Varga, Pongor [15]). If one is interested to create layer models that can be run on different simulation kernels, then some simulator specific layer between the models and the kernel must be inserted, though this will not be treated in this work.

The following subsections will describe the purpose of each layer in detail. Many of the models will consist of a finite (yet dynamically changing) set of states, which might change over time. Until stated otherwise, the described layers are assumed to use the above set of standard signals.

3.4 Workload Generator

At the highest layer in the framework, a workload generator is associated with each node set. This workload generator creates user sessions according to the desired statistical distribution. This can be done in two ways:

- The workload generator is associated with a static node set. In this case, a currently free member of the static node set is chosen to host the next user session. As an example, a company network can be taken, where employees arrive in the morning and start generating network requests on their workstations.
- The workload generator is associated with a dynamic node set: In this case, nodes are created according to their node templates and are connected to a member of their associated node sets (dial-in nodes). Here, as an example, a mobile communication network can be taken, where users arrive according to some arrival rate, and start generating workload using their mobile terminal equipment.

Generating user sessions can either be done synchronously or asynchronously to its next lower layer:

- Synchronous: The time to create the next user session instance is computed only, if the next lower layer has ended the session. This results in a sequence of none-overlapping sessions.

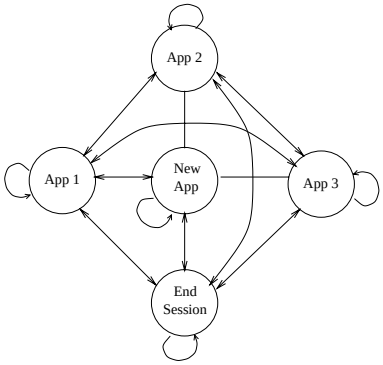

Figure 2. Session layer model.

- Asynchronous: The time to create the next user session instance is computed right after the previous user session instance has been created. In this case, sessions can overlap and competition for the available nodes might occur.

Workload generators thus must be able to host one or more user sessions. The workload generator instances are created at simulation start. Each user session instance is then created and linked to its workload generator instance at runtime.

3.5 Session Layer

Models at this layer have the following tasks:

- Start applications.
- Choose the application to use.
- End session.

Note that it is only the application to use, not the service that is chosen. Also, the number of possible states of models at this layer may vary, as in principle, many applications can be started and stopped again. Figure 2 shows an example for such a session layer model.

Once, an application is chosen, the next lower layer starts services of this application. Starting and choosing applications can again be done either synchronously or asynchronously.

Session layer models are plugged directly on top of application layer models. Note that session layer models can be very simple dummy models that just start an application (which in turn just chooses and starts a command sequence). This way, each TE/user can create workload with very small effort at a low level of detail.

3.6 Application Layer

Application models define, how the user interacts with the chosen application. Basically, the user can perform one of the following interactions:

- Start a new command sequence.
- Choose a running command sequence.
- Stop the application.

If the application is stopped, the user session instance above must start another application or choose another one. The generation or selection of command sequences can again be done synchronously or asynchronously to the user commands at the next lower layer.

Application layer models are plugged on top of command layer models. Note that application layer models can be very simple dummies, having nothing more to do than starting a command sequence.

3.7 Command Layer

At this layer, sequences of commands for lower level services are generated. This layer consists of three independent sub-models.

3.7.1 User Commands

Here, services are started or running services are selected, and sequences of commands are issued for running services. Commands for running services denote the changing of parameters like the size or color depth of a running video conference.

Amongst the possible commands are:

- Start a service
- Change service parameters.
- Stop the service.

If a service is started, it generates workload requests and passes them down to the resource layer. The observed quality of service (QoS) is part of the service and can be obtained by the next higher level. Services can stop themselves after delivering the result (for example HTTP). If a service is stopped, the command layer model must take over again and generate the next command.

User commands can be simple dummy models, which either do nothing or just start a service or stop it after some time.

3.7.2 QoS Checks

As long as a service is running, it delivers QoS descriptions. For each QoS description, a QoS level can be defined. QoS checks compare the observed QoS level to the requested QoS level. QoS checks can be performed as single or periodic checks:

- Single: The check is performed only once, after a time out occurred. If the service has not been completed, the appropriate QoS action is triggered (for example the requested web document has not been downloaded)
- Periodic. The check is performed periodically, until the service stops.

3.7.3 QoS Actions

If QoS checks fail, an appropriate QoS action is triggered and performed. QoS actions and user actions are at the same level. They can start or stop services, or change parameters for running services.

3.8 Service Layer

Service layer models consist of parameterized traffic generators (Jagerman, Melamed, Willinger [10], Hlavacs, Kotsis, Steinkellner [9]). Some parameters can be changed by higher level models and influence the QoS level that the higher level model chooses. Other parameters are fixed and describe a situation that can not be changed by users, like the distribution of file sizes at a web site.

Services can stop themselves, it they have performed their task (like delivered the web document, played a video). Services can also be represented by very simple models, which just create low detail workload by using simple fluid models or renewal models like Poisson arrivals.

In addition to standard signals, the service layer will send the following signals:

- SIG_SEND: Sends n bytes to the next lower layer (resource layer)
- SIG_REQ: The service requests n seconds or n bytes from a resource (CPU, main memory,I/O).
- SIG_RECV: Tells the next higher layer model instance that the service delivers results.

3.9 Resource Layer

Resource layer models describe physical node resources like networks (TCP/IP, ATM), CPU queues, memory pages and I/O resources like disks. Each of these models must be implemented at least as a dummy resource. There is only one resource layer model instance per workload generator instance. The network resource model will in general be attached to another network resource model within the network.

When using existing simulation kernels like ns, some of these resources will only be entry points to the network simulator kernel.

The following signals will be sent by the resource layer:

- SIG_SEND: Sends n bytes to an attached resource at resource layer.
- SIG_RECV: Tells the next upper layer model instance that the resource is delivering

4 Modeling Methodology

The previous sections describe a framework of model classes and how to glue these model classes together. Yet it says nothing about the type of models to use. Its strongest suggestion is that each model is always in one well defined state, where the number of possible states may vary dynamically. As model types, any standard model like Markov Chains of order 1 or higher, or other stochastic processes would fit.

Modeling will be divided into two parts:

1. Modeling user behavior: In this part, explicit models for users will be derived, whenever the data to do so is available. If this is not the case, then the observed workload will be described at the highest possible layer (e.g. the stream of web requests, the stream of observed LAN packets, the stream of observed ATM packets). Also, layers for which no data is available will be replaced by dummy layers.
2. Modeling service workload: This model will generate the actual workload and will put it into a resource layer model instance.

Creating realistic user models will require the data to do so. For example, data derived from a WWW proxy log will allow to generate a user model starting applications and starting HTTP services depending on previous services. It will not allow to construct QoS checks, as this is not recorded in the proxy log. Here, simple heuristics must be inserted.

5 Case Study: Modeling HTTP Requests

As an example for the layered approach, in this section, user models for creating HTTP requests will be presented. These user models will then drive a network simulator to derive performance measures.

Let us assume that the simulation goal is to derive performance figures for a hypothetical computer science lab, an institution is planning to install. The question to answer is, how many workstations to buy, and how to dimensionate the network connecting the lab with the internet. Students arrive according to some given exponentially distributed interarrival time and try to find a free workstation. If no workstation is available, the students will immediately go away.

Based on the collected data, user sessions are classified into several types. Users arriving at some arrival rate would then be represented by one instance of a session type. For each session type, one browser would be started, producing a stream of file requests within each session.

This way, as opposed to directly modeling streams of web requests which requires detailed knowledge on the statistical properties, analysts using the simulator would be able to alter the workload to be created by just changing the session arrival rate, or by additionally changing the mixture of session types.

5.1 Data Collection Environment

Since early 1999, a Squid proxy server has been installed in one the computer science labs of the University of Vienna. As a standard web browser, Netscape 4.08 has been provided. For all web browsers, the local cache has been switched off and locked, such that users are not able to switch it on again. Thus, all web requests are led through the central proxy server. The data under study has been collected in the interval from February 23rd to May 6th, leading to a total of 51516 downloads. The downloads have then been grouped together by creating user sessions.

5.2 User Model Details

For finding session limits, the download interarrival time between 2 downloads from the same IP address has been used. The empirical cumulative distribution function of these differences revealed that over 95% have been less than 500 seconds. Moreover, the histogram showed a significant flattening at about 1800 seconds. A new session thus is said to start, if the difference is either larger than 1800 seconds, or larger than 500 seconds plus a change in the web server address serving the download.

By using this strategy, 224 sessions were identified, from these, 196 had more than 15 downloads, which we considered a minimum to deliver meaningful data. Moreover, the interarrival time between sessions has been derived, the histogram is shown in Figure 3. It clearly shows a close (though not exact) relationship to an exponential distribution with a mean of 1278.246 seconds.

The MIME types of the downloaded file were grouped together, forming the five groups stated in Table 1. Each

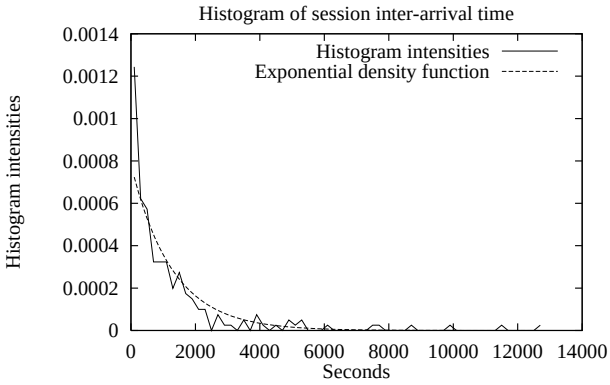

Figure 3. Histogram of session inter-arrival time.

T_1	T_2	T_3	T_4	T_5
appl.	audio/video	image	text	unknown

Table 1. MIME type groups.

session then was represented by its mixture of downloaded filetypes in percent, yielding 196 5-dimensional vectors. The five dimensions then were reduced to four by using principal component analysis, the percent of variance explained by the principal components can be seen in Table 2.

Furthermore, the vectors then were classified by using the k-means clustering algorithm for different numbers of clusters. Figure 4 shows the sum of squared intra-cluster errors for different numbers of clusters. After 5 or more clusters, the relative decrease in the errors is clearly getting smaller, and therefore, few can be gained by further increasing the number of clusters. Thus, 5 has been selected as the number of clusters to use. Table 3 shows the percentage of sessions within each cluster, yielding also the probability of creating a session belonging to this cluster.

For each cluster, a Markov chain of order one was created, representing the sequence of downloading files of one of the five MIME type groups. Two additional states have been added to these Markov chains, one for session start, and one for session end. The session duration thus is given implicitly by the probabilities to enter the end-state.

Furthermore, for each of the clusters and each MIME

V_1	V_2	V_3	V_4	V_5
38.4	28.14	21.93	11.56	7.8e-05

Table 2. Percent of variance explained by the principal components.

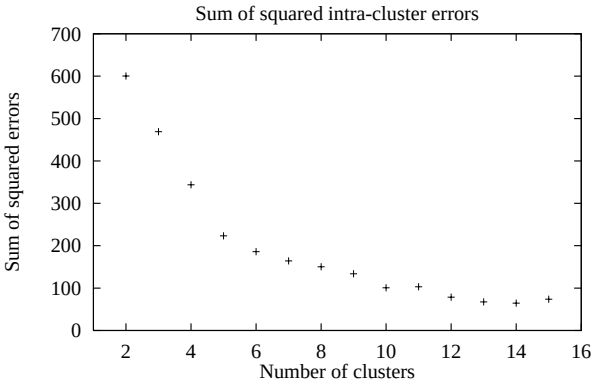

Figure 4. Sum of squared intra-cluster errors.

C_1	C_2	C_3	C_4	C_5
7.7	59.1	1.0	28.1	4.1

Table 3. Percent of sessions contained in each cluster.

type group, an empirical cumulative distribution function of the observed file sizes was created. This has also been done for the interarrival time of download requests within the sessions.

5.3 Simulation Results

A prototypical discrete event simulator has been implemented in C++, using the framework described in Section 3. Figure 5 shows the simulator C++ classes.

The set of classes contains base classes for each of the layers, all derived from a single *Eventhandler* class. These base classes serve as dummy layers, such that after receiving the signal SIG_START, they create one instance of the next sub-layer and send the signals SIG_INIT and SIG_START to them. For modeling the user sessions representing the collected data, additional classes have been derived from the layer base classes, implementing the model types of choice.

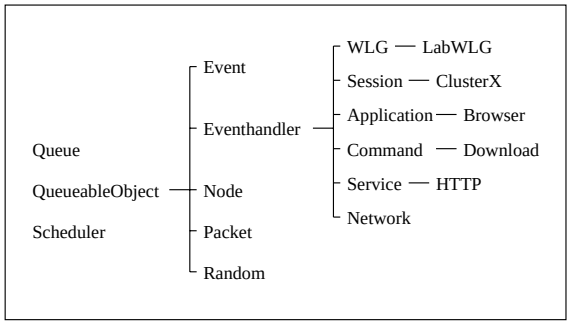

Figure 5. Simulator classes.

The process creating user sessions according to the chosen arrival rate, as well as choosing the next session type, was placed into a class derived from the workload generator base class. The session layer then was represented by a dummy model, simply creating and starting a model derived from the application layer base class. There, the interarrival time for file requests, together with the desired file types, are created. Finally, inside the command layer model, the file sizes for the chosen session types and file types are computed for each request. The service layer model again serves as a dummy layer by just inserting arriving data into the network model implemented at the resource layer. There, the data is put into a queue and removed after the time span necessary to be transported on a 2 Mbit network.

As performance figures, the probability of finding no free workstation, and the utilization of a 2 Mbit network line were computed. The number of students arriving per hour was chosen to be in the interval from 1 to 120, in incremental steps of 5. For each of these arrival rates, several simulation runs were carried out, each creating 300 user sessions and yielding an estimate for the parameter under study (rejection probability, network utilization). This was done, until the ratio of the 95 percent confidence interval length to the mean of these estimates was below 0.10, but with a maximum of 50 runs. This means that if the limit was met within 50 runs, with a probability of 95 percent, the error of the computed performance measure is below 5 percent. Computation was carried out on the 5 node Beowulf cluster installed at the Institute of Physical and Theoretical Chemistry at the Technical University of Vienna, each node being equipped with two Pentium II 350 MHz processors. Using a naive parallelization strategy, the computing time was less than 1.5 hours, which could be advanced further by applying some load balancing technique. Figures 6 and 7 show the simulation results. It can be clearly seen that it is necessary to provide between at least 10 to 20 workstations. On the other hand, the network bandwidth of 2 Mbit/sec seems to be sufficient to transport the produced web traffic.

The analyst can now easily experiment with different scenarios, e.g. by varying the mix of sessions and study the effects on network performance.

6 Conclusion

In this paper, a layered framework for user behavior oriented workload modeling has been presented. The layered approach supports the characterization of the workload at the desired level of detail, ranging from an explicit implementation of user behavior at the top level to a low level description of the resulting load at the resource layer.

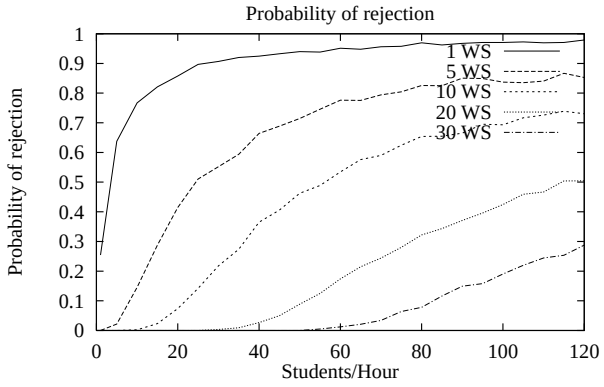

Figure 6. Probability of rejection depending on student arrival rate.

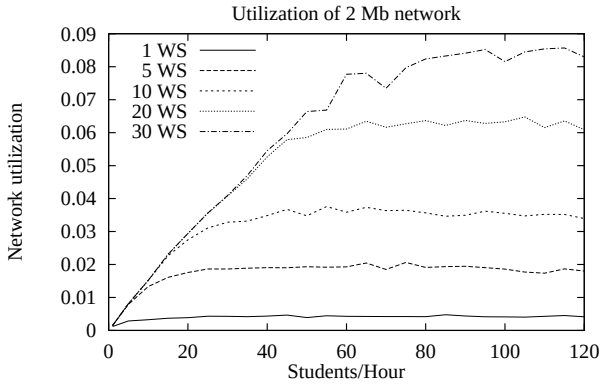

Figure 7. Utilization of 2 Mbit network depending on student arrival rate.

Plugging workload model components into the different layers of the framework supports modeling flexibility in that different types of models, including pre-defined as well as self-defined models, can be included in the framework. Analysts using a simulator of this kind can then easily choose between the various model types to drive simulation runs. As a prototypical realization of the framework, a simulator reflecting the layered structure has been created in C++ and used to derive performance figures for a hypothetical computer environment. In this case study, we have demonstrated, that the analyst can easily perform what-if scenarios by changing a high-level workload parameter, like the inter-session arrival time, and observe the effects of this change on lower levels, e.g. the actual traffic load imposed to the network.

Current and future work focuses on the implementation of the proposed framework, providing a library of reusable, parameterizable model classes and its integration into popular network traffic simulators (e.g. OPNET [12] and ns [2]).

References

[1] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song. Parsec: A parallel simulation environment for complex systems. *Computer*, 31(10):77–85, 1998.

[2] S. Bajaj. Improving simulation for network research. Technical Report 99-702, University of Southern California, Mar. 1999.

[3] M. Calzarossa, G. Haring, G. Kotsis, A. Merlo, and D. Tessera. A hierarchical approach to workload characterization for parallel systems. In B. Hertzberger and G. Serazzi, editors, *High Performance Computing and Networking, LNCS vol. 919*, pages 102–109. Springer, 1995.

[4] M. Calzarossa, G. Haring, and G. Serazzi. Workload modeling for computer networks. In U. Kastens and F. Ramming, editors, *Architektur und Betrieb von Rechensystemen*, pages 324–339, Muenchen, 1988. Springer-Verlag.

[5] M. Calzarossa, R. Marie, and K. Trivedi. System performance with user behavior graphs. *Performance Evaluation*, 11:155–164, 1990.

[6] L. Catledge and J. Pitkow. Characterizing browsing strategies in the world-widw web. *Computer Networks and ISDN Systems*, 26(6):1065–1073, 1995.

[7] C. Chen. Structuring and visualizing the www by generalized similarity analysis. In *Proceedings of Hypertext '97*, 1997.

[8] M. Crovella. Generating representative web workloads for network and server performance evaluation. *Performance Evaluation Review*, 26(1):151–160, 1998.

[9] H. Hlavacs, G. Kotsis, and C. Steinkellner. Traffic source modeling. Technical Report TR-99101, Institute for Applied Computer Science and Information Systems, 1999.

[10] D. Jagerman, B. Melamed, and W. Willinger. Stochastic modeling of traffic processes. In J. Dshalalow, editor, *Frontiers in Queuing:Models, Methods and Problems*. CRC Press, 1996.

[11] D. Menasce, V. Almeida, R. Fonseca, and M. Mendes. Resource management policies for e-commerce servers. In *The 2nd Workshop on Internet Server Performance (WISP 99)*, 1999.

[12] MIL3. Opnet, www.mil3.com.

[13] V. Noethe. User behaviour at system command language level. *Computer Performance*, 3(1):5–9, 1982.

[14] S. Raghavan and P. Joseph. Workload models for multiwindow distributed environments. In *Quantitative Evaluation of Computing and Communication Systems*, Heidelberg, 1995. Springer-Verlag.

[15] A. Varga and G. Pongor. Flexible topology description language for simulation programs. In *Proceedings of the 9th European Simulation Symposium (ESS'97)*, pages 225–229, 1997.

[16] T. Yan, M. Jacobsen, H. Garcia-Molina, and D. Umeshwar. From user access patterns to dynamic hypertext linking. *Computer Networks and ISDN Systems*, 28(7-11):1007, 1996.