WORKLOAD GENERATION BY MODELING USER BEHAVIOR*

Helmut Hlavacs & Ewald Hotop & Gabriele Kotsis
Institute for Computer Science and Business Informatics
University of Vienna, Austria
{hlavacs, hotop, kotsis}@ani.univie.ac.at

ABSTRACT

In this paper, a user-behavior oriented workload generator for OPNET is presented. In this novel approach, both packet generation and user interaction with applications and services that cause the network traffic are modeled. This way, the analyst can model traffic and construct a workload at a higher level of abstraction. The user behavior is represented by a user profile, defining a hierarchy of independent processes that exchange standardized messages with each other. Generally, any type of stochastic process can model such processes. In the current implementation, processes are modeled by finite state machines whose states denote the actions a user can perform. A collection of text files defining a process hierarchy is called a user profile. User profiles are stored in a user profile database that is a sub-tree of a Windows or Unix file system. Due to the hierarchical profile structure, it is possible to define user behavior models in a modular manner. This way, new user profiles can easily be constructed by reusing existing (sub-) profiles.

INTRODUCTION

Future computer systems and network components will face a mixture of different workload types, varying from simple to heavy. The upcoming digitalization of multimedia will impose especially tight constraints on the quality of service (QoS) that the computing and network resources have to provide. If those requirements are not met, users will be dissatisfied, which may lead to the loss of customers or may render a certain computer system worthless. Thus, before actually implementing a new network, analysts should 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, requiring reliable, and prototypical models for the generation of artificial workloads. Low-level workload 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 time.

In this paper, a workload generator simulating the behavior of users of interactive computer systems is presented. It is meant for the simulation of network traffic, stand-alone computer systems and any mixture thereof.

RELATED WORK

Models for generating realistic computer and network workload have been under study for many years. Many models are represented by stochastic processes ([8], [6]) 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 transferred over the network.

User behavior models try to catch the sequence of user interactions at a higher level. Such models will in general be hierarchical, as the workload generated at a higher level will result in a stream of workload requests at a lower level. Examples for hierarchical user models can be found in [12] and [3]. Another popular method for catching inter-command dependencies is to apply Markov Chains for user behavior modeling ([9], [2], [1], [4]). More static approaches can be found in [11] and [13], where users are described by *n*-dimensional vectors. In [5], users are represented by so-called "user equivalents", which are simple two-level heavy tailed On-Off processes.

THE BISANTE WORKLOAD GENERATOR

The workload of a computer system or network can be described at various levels, ranging from a user or

* This work was performed in the BISANTE (Broadband Integrated Satellite Network Traffic Evaluations) Esprit project supported by the EC under research grant EP28425. See http://www.bisante.org

application oriented view to a resource or system oriented view. Depending on the objectives of the evaluation study, the analyst should be able to choose the appropriate level of characterization. However, most existing workload generators only support a characterization at the lower levels (in terms of resource requirements, packet arrival rates, etc.), not allowing for a description and representation of the underlying user behavior.

The goal of the BISANTE workload generator modeling framework ([7]) is to find sets of layered models that can be plugged into each other to represent some typical user behavior that has been either observed or that is projected to be seen in the future. As the user behavior is modeled by a tree of independent processes communicating with each other by sending standardized signals, any type of stochastic process like Petri Nets or Markov Chains may be used. The currently implemented model types consist of different versions of finite state machines (FSMs) specialized for certain tasks like reacting to QoS or generating packets.

The model implementation, however, is separated from the stochastic model description, i.e., the states an FSM of a particular user profile at a certain level can enter, the distribution for the time between two state changes, and which sub-processes it can create. This data is stored in the *BISANTE user behavior database*, a sub-tree of a Windows or Unix file system. Different user profiles are put into different sub-trees of this database, where they are represented by text files describing each node of the FSM hierarchy.

The analyst using the BISANTE workload generator first selects one or several user profiles from the database and then attaches the selected user profiles to nodes in his scenario. At simulation start, the workload generator implementation creates the processes inside the scenario nodes, each process then reading in its description from the user behavior database.

USER PROFILES

The behavior of users is represented by a tree of text files, where each text file describes a stochastic process that at run-time may be created by its parent and may itself create child processes as well. The tree itself then is called a *user profile*.

Figure 1: Hierarchical user profile.

Figure 1 shows such a tree. The text files describing this situation are shown in Table 2. Here it is assumed that the profile database is stored in directory "PD" somewhere in the file system.

Filename	Description
PD\wlg\wlg.txt	Points at user profile us1 with file name "session.txt".
PD\profiles\us1\session.txt	Description of session us1\session. Contains session-length pdf and type of applications that may be started.
PD\profiles\us1\app1\app1.txt	Description of application us1\app1. Contains commands that may be started.
PD\profiles\us1\app1\cmd.txt	Description of command us1\app1\cmd. Contains information about how to react to observed QoS and about types of services to be started.
PD\profiles\us1\app1\srvc1.txt	Description of service us1\app1\srvc1. Contains information on which server to connect to, how many requests/packets to send to the server, packet inter-arrival time pdf and packet-size pdf.
PD\profiles\us1\app1\srvc2.txt	...
PD\profiles\us1\app1\srvc3.txt	...

Table 2: Example user profile.

As a user behavior profile may consist of several different text files stored in succeeding directories, it may seem that it is generally complicated to construct or use the BISANTE user behavior profiles. However there are several reasons why this is not necessarily the case:

- User profiles may be very simple. For example, a Workload Generator may immediately create a service that is no more than a flexible packet generator. This scheme would provide similar functionality as is provided in regular packet generators.
- One task of BISANTE is to create a representative user behavior database. This means that user profiles for various applications and protocols will be provided. Analysts may simply use the provided profiles, or may create new profiles by copying and adapting appropriate sub-profiles.
- As the user behavior and all stochastic properties are described in simple text-files, automatic generation of user profiles is an easy task.
- A planned graphical user interface will provide quick profile overview and manipulation.

The idea behind the BISANTE workload generator is to use very simple models for packet generation only *when feasible*, but to be able to create user behavior profiles of arbitrary complexity *whenever needed*. Of course, any mixture of simple and complex models is possible.

In a simulation scenario, one or several nodes may point to a particular workload generator described in the user behavior database. In OPNET this is done by setting the node attribute "BI_WorkloadGeneratorName" defined in the BISANTE process model to the name of the appropriate text-file. An example can be seen in Figure 3.

Figure 3: Binding a WLG to a node.

In the file describing the workload generator, the locations and names of user behavior profiles that may be created at run-time are stored.

IMPLEMENTATION

The model types are implemented by C++ classes. The basic FSM functionality is implemented in the *Module* class; all other classes are then derived from Module.

Type	Description
event handler	Super-class of all process types. Provides functionality to interact with simulator. New process types must be derived from this class.
module	Basic finite state machine class.
workload generator	Derived from module. Exactly one workload generator is attached to each node in the scenario. The WLG will then create sessions on its own node or on other nodes in the scenario.
command	Derived from module, a command is meant to interact with services (i.e., the packet generators) to evaluate and react to the observed QoS. It may also change parameters of its child processes.
service	Derived from command, a service is a flexible packet generator. It may open connections to servers and may send packets/requests to them. It may additionally create QoS reports.

Table 4: The main C++ classes.

All C++ code is stored in the libraries "bisante.lib" (Windows) or "bisante.a" (Unix). In order to make OPNET link the library to the simulation executable at simulation start, the library name and location must also be put into the *OPNET environment database*.

THE FSM STATES

As was said before, the currently implemented processes are basically finite state machines. The states here denote the actions or *events* that the machine can perform. When being in state s0 (s0 is the last executed event) at time t and choosing event s1 to be performed next, s1 is scheduled to be performed at time $t+dt$, where $dt \geq 0$ is a random number attached either to s1 directly, or to the transition $s0 \rightarrow s1$ if this transition is stated explicitly. If no random number is provided for s1 or the transition $s0 \rightarrow s1$, dt is set to 0. At time $t+dt$, s1 is performed and the next event s2 is chosen and scheduled. If no new event is found, nothing is scheduled and the FSM waits

for signals to arrive from its parent or any child processes.

However, any such FSM is also an On-Off process, a process that is either in the super-states On or Off, here called Started or Stopped. The scheme described above will only be performed if the process is in super-state Started. If it is in super-state Stopped it may not schedule new events or react to incoming signals. This On-Off process scheme is also reflected in the OPNET process model used, shown inFigure 5.

Figure 5: The process model.

Choosing the next event to be executed may also be done upon the arrival of a *signal*. This is the basic mechanism for processes to communicate with their parents or children. A signal sent from parent to child is a *command* (not to be confused with the C++ class Command), i.e., it must be executed immediately by the receiver. Signals sent from children to their parents are called *indicators*, i.e., they flag a change of the child's state and the parent may or may not react to this signal. If a process is to react to a signal, this must be stated explicitly in the profile.

Table 6 contains the events that may be scheduled by a standard Module for itself (starting by "EVENT_"), and the signals it may send or react to (commands start with "CMD_", indicators start with "IND_"). Additionally, "(s)" denotes the fact that this process type may send this signal, while "(r)" denotes the fact that this process type may react to this signal by choosing a new event from its profile.

Table 7 contains the events and signals that additionally may be scheduled or processed by Commands.Table 8 contains the events and signals that additionally may be scheduled or processed by Services.

Event/Signal	Description
EVENT_NEW_SL	Create a new child process.
EVENT_STOP	Module stops itself.
EVENT_END	Module ends itself.
EVENT_STOPCHILD	Sending a CMD_STOP signal stops a child.
EVENT_ENDCHILD	Sending a CMD_END signal ends a child.
CMD_INIT (s)	A child must initialize itself (after creation).
CMD_START (s/r)	A child must enter the Started super-state.
IND_STOPPED (s/r)	Sent to the parent when entering the Stopped super-state.
IND_ENDED (s/r)	Sent to the parent when entering the Ended super-state.

Table 6: Module events and signals.

Event/Signal	Description
EVENT_REQ	Send a CMD_REQ to a child, which immediately will change certain parameters.
CMD_REQ (s)	Forces a child to change parameters, e.g., the packet inter-arrival time.
IND_RECV (r)	A QoS report is sent from a child, holding a QoS class (a string) and a QoS level (a floating-point number). For example, the class may be "bitrate", and the level may denote 1 Mbit / s.

Table 7: Additional Command events and signals.

Event/Signal	Description
EVENT_NEW_PACKET	Send a new packet or request to the partner Service.
EVENT_QOSREPORT	Generate a QoS report.
CMD_REQ (r)	Change parameters.
IND_RECV (s)	Send a QoS report to parent.

Table 8: Additional Service events and signals.

As was said before, the Service class is derived from the Command class, i.e., a Service may also act as a Command. As Commands are meant for controlling Services, a Service may also control sub-services. For example, a TCP-based control channel may control one

or several sub-channels carrying audio or video data over UDP connections.

CASE STUDY: HTTP REQUESTS

As an example for the BISANTE workload generator, in this section, a scenario of web users downloading documents from the web will be simulated. Let the simulation goal be to derive performance figures for a hypothetical computer science lab that an institution is planning to install. The question to answer is, how many workstations to buy, and how to dimension the network connecting the lab with the Internet? Students arrive according to some given exponentially distributed arrival rate and try to find a free workstation. If no workstation is available, the students will immediately go away.

Since early 1999, a Squid proxy server has been installed in one of the computer science labs of the University of Vienna. All web download requests from inside the lab were led through this proxy server. The logged data (in total 51516 downloads) were used to derive five user classes. Details of this analysis can be found in [7].

The above-described user classes where then described by user profiles stored in the BISANTE userbehavior database shown in Table 9.

PD\wlg\bisante\http\wlg.txt
PD\wlg\bisante\http\server.txt
PD\wlg\bisante\http\client.txt
PD\profiles\bisante\http\server\server.txt
PD\profiles\bisante\http\class1\session.txt
PD\profiles\bisante\http\class1\browser.txt
PD\profiles\bisante\http\class1\http_get.txt
PD\profiles\bisante\http\class2\session.txt
PD\profiles\bisante\http\class2\browser.txt
PD\profiles\bisante\http\class2\http_get.txt
...

Table 9: Web user profile files.

User sessions arriving at some arrival rate are then represented by one instance of the five classes chosen at random. For each instance, one browser is started, producing a stream of file requests within each session. This way, as opposed to directly modeling streams of web requests that requires detailed knowledge of 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.

SIMULATION RESULTS

The computer science lab has beenmodeled in various scenarios, containing 1, 5, 10, 20 or 30 client computers. On each client computer, the Workload Generator described in file "PD\wlg\bisante\http\client" was placed by setting the node attribute "BI_WorkloadGeneratorName" to "bisante\http\client". Additionally, one node not connected to the network was placed into the topology to create arriving student sessions. On this node, the node attribute was set to "bisante\http\wlg". On the server node, the node attribute was set to "bisante\http\server".

Additionally, it was assumed that students arrive at a rate of $1 \leq \lambda \leq 120$ students per hour, with increments of 5. If a student finds a free client computer, he immediately starts to download files from the Web; otherwise the student is rejected and leaves the facility. For each λ, the scenario was simulated for 12 hours. The simulations were carried out on three workstations in parallel, simulation times varied from 100 s to over 14,000 s. At a rate of up to 11,000 events/s, each run produced between a few thousands and over 100,000,000 events.

Figure 10: Probability of student rejection.

Figure 10 shows the probability of a student finding no free workstation.

Figure 11: Downloads per hour.

Figure 11 shows the number of downloads per hour. The mean throughput from the Web to the lab can be seen in Figure 12.

Figure 12: Mean network throughput.

CONCLUSIONS

In this paper, the OPNET implementation of the BISANTE workload generator has been introduced. It consists of an open framework of hierarchies of stochastic processes that communicate with each other by sending signals. A similar hierarchy of text files that contain the description of the processes used describes the process hierarchy. The text files are stored in the user behavior database, a sub-tree of a standard Windows or Unix file system, reflecting the hierarchical user behavior profile. The currently implemented processes act as finite state machines, though any kind of stochastic process may be used instead, as long as it is derived from the C++ class Event Handler and adheres to the signaling conventions.

The currently available OPNET version has been used to simulate hypothetical computer science lab scenarios. 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 on the network.

FUTURE WORK

Current work focuses on providing a mature and usable product for OPNET and NS, the Berkeley Network Simulator. Also, several different sources for network traffic are analyzed and modeled by user behavior profiles.

Future work will also include the development of a simple graphical user interface for the user behavior database.

REFERENCES

[1] M. Calzarossa, G. Haring, G. Serazzi, *Workload modeling for computer networks*. In "Architektur und Betrieb von Rechensystemen" (U. Kastens and F. Ramming, Eds.), Springer-Verlag, Berlin 1988, pp. 324—339.

[2] M. Calzarossa, R. Marie, K. Trivedi, *System performance with user behavior graphs.* Performance Evaluation 11 (1990) , pp. 155—164.

[3] M. Calzarossa et al., A hierarchical approach to workload characterization for parallel systems. In "High Performance Computing and Networking" (B. Hertzberger and G. Serazzi, eds.). Springer LNCS, Berlin 1995, pp. 102—109.

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

[5] M. Crovella, *Generating representative web workloads for network and server performance evaluation.* Performance Evaluation Review 26-1 (1998), pp. 151—160.

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

[7] H. Hlavacs, G. Kotsis, *Modeling User Behavior: A Layered Approach.* In "MASCOTS'99", IEEE Computer Society, Los Alamitos, California 1999, pp. 218—225.

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

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

[10] MIL3, OPNET, http://www.opnet.com

[11] V. Noethe, *User behaviour at system command language level*. Computer Performance 3-1 (1982), pp. 5—9.

[12] S. Raghavan, P. Joseph, Workload models for multi-window distributed environments. In "Quantitative Evaluation of Computing and Communication Systems", Springer-Verlag, Berlin 1995.

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