A Platform for Real-Time Visualization and Interactive Simulation of Large Multimedia Networks

M.C. Chan, G. Pacifici*, and R. Stadler

Center for Telecommunications Research
Columbia University, New York, NY 10027
{mcchan, stadler}@ctr.columbia.edu

*IBM T.J. Watson Research Center
Hawthorne, NY 10532
giovanni@watson.ibm.com

Abstract

We present a platform for developing and evaluating control systems for emerging multimedia networks. The platform allows us to closely approximate the functional and dynamic behavior of network control systems. By providing support for real-time visualization and interactive emulation, it can be used to study multimedia networks in various scenarios, such as different load patterns, network sizes and management operations. The current implementation runs on a KSR-1 and an SP2 parallel processor, which are connected to a graphics workstation via ATM links.

1. Introduction

In order to provide multimedia services, such as interactive television, video conferencing, and access to digital libraries, new control and management systems for high-speed networks are required. Testbeds, recently set up to address this issue, have demonstrated the feasibility of providing high-speed connectivity and multimedia communication among a small group of users.

However, it is difficult to use these testbeds as a tool for developing new control and management systems for high-speed networks. The difficulties stem from the prohibitive cost of instrumenting the testbeds with traffic generators and monitoring capabilities for high-speed traffic load. Also, changing the existing functionality or configuration is generally time consuming. Furthermore, it is hard to use these testbeds to determine whether their control systems scale well to networks of larger size.

An alternative approach to study network design problems is using simulators [1,3,9,10,11]. The advantage of simulation environments over testbeds is that they allow for easy monitoring of the system state and rapid modification of the functionality of the simulated system. While these simulators are suited for studying the behavior and performance of a specific controller or an isolated subsystem, they are not powerful enough to investigate the collection of mechanisms that compose the control and management architecture of a multimedia network.

In order to study multimedia network architectures, an environment with the following properties is needed:

This research was supported by the Department of the Air Force under contract F30602-94-C-0150. It was conducted using the resources of the Cornell Theory Center.

(1) Real-time visualization: Due to the complexity of multimedia network control systems [13], it is necessary to visualize the evolution of the network state in real-time and to display different abstractions of the state. (2) Interactivity [8,15]: The operation of a network control system depends on a very large number of control, configuration, and environment parameters. Therefore, the capability of changing many of these during run time is needed, in order to execute what-if scenarios in a flexible way. (3) Large computational resources with low-latency interaction: In order to emulate a network architecture in real-time, we need the computational resources that would be available in the emulated target system and also the ability for fast interaction. We think this requirement can best be met by using a massively parallel machine.

In this paper, we present an emulation platform, which exhibits these properties, i.e., real-time visualization, interactivity, and a large amount of computational resources with fast interaction among objects. The emulation platform runs on two types of parallel computers, the KSR-1 and the IBM SP2. The front end is implemented on an SGI graphics workstation. This platform is currently used in various projects in our laboratory which are aimed at developing and evaluating network architectures. The paper is organized as follows. Section 2 describes the design of the emulation platform. Section 3 gives implementation details and the physical platform involved. Section 4 discusses our experience in using the emulation platform to develop an architecture for managing multimedia network services.

2. The Emulation Platform

The emulation platform consists of four building blocks: *parallel simulation kernel*, *emulation support*, *real-time visualization and interactive control*, and *emulated system* (Figure 1). The emulated system and the emulation support modules consist of a set of objects that communicate by exchanging messages, using send and receive functions provided by the simulation kernel. The simulation kernel controls the execution of these objects and insures that messages are processed in the correct order. In order to support real-time visualization and interactive control of the emulated system, the kernel controls the progression of the simulation time, constraining it by the progression of the processor time. (Other parallel simulation kernels discussed in the literature include [2,12,14].)

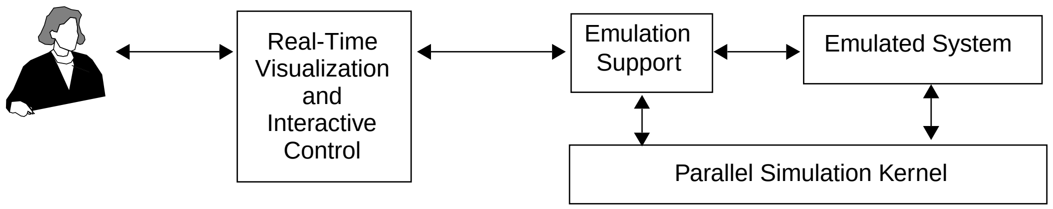

Real-Time Visualization and Interactive Control
Emulation Support
Emulated System
Parallel Simulation Kernel

Figure 1: Building blocks of the interactive emulation platform.

Objects that interact with the parallel simulation kernel require little knowledge about the kernel--mainly how to send and receive messages. Therefore, the design of the emulated system follows the same rules as the design of network controllers that run on a real network platform. The major difference is in how the interaction among objects/controllers is realized. In the emulated system, interaction is performed by the simulation kernel. In a broadband network environment, for example, the exchange of messages is provided by a signalling system.

The module for real-time visualization and interactive control contains a graphical interface which provides 3D visual abstractions of the system state. The state of the emulated system is continuously refreshed, which allows a user to follow the system dynamics on the interface. The user is able to control the behavior of the simulated system by changing parameters through the interface.

The emulation support module coordinates the exchange of control and monitoring messages between the module for real-time visualization and interactive control and the emulated system. It reads the states of the emulated system, and performs filtering and abstraction operations before making the information available for visualization. Control information from the user is mapped to a set of control parameters that are interpreted by the emulated system.

The design of the emulation platform along the four building blocks described above serves the purpose of portability and re-usability. The platform runs on two types of supercomputer with shared memory and distributed memory architectures, respectively. The two implementations differ in the emulation support and the simulation kernel modules. The emulated system module needs only minor changes when porting the software from one computer to the other; the module for real-time visualization and interactive control needs no modification. Also, we use our platform in several projects where network architectures are developed and evaluated [4,13]. Experimenting with a different architecture requires mainly that we load a different emulated system module. Furthermore, we use the interactive interface module for visualizing the systems we are emulating, as well as for developing interfaces for network operators [5]. In the following, we discuss the design and implementation of each of the four building blocks.

2.1 Parallel Simulation Kernel

The parallel simulation kernel implements the paradigm of parallel discrete event simulation (PDES) [7] and supports some real-time capabilities for interactive emulation. The simulation kernel controls the execution of simulation objects and routes events from one object to another. Objects that require services from the kernel are defined as a subclass of `SimObject`. Each `SimObject` (or any subclass of this object) has an `execEvent` method with an input parameter of type `Event`. Attributes of this type include a time-stamp indicating when the event can be processed, object identifiers of source and destination, and an event-type tag. An object generates an event by invoking the `scheduleEvent` method which has an output parameter of type `Event`. The simulation kernel receives this event, routes it to the destination, and invokes the `execEvent` method of the destination object.

Figure 2 shows the realization of the parallel simulation kernel. It is implemented as a set of local simulation kernels, each running on a separate processor. A local simulation kernel controls the execution of objects allocated to its processor. Routing of events is performed using a copy of a global object allocation table which is set up during the initialization phase of the simulation.

The simulation kernel controls object execution using a global time T_G. It delivers an event to an object for execution only when the time-stamp of the event is smaller or equal to T_G. Otherwise, the event is buffered in the event queue. T_G is computed as the minimum of T_S and T_W, which are explained below.

T_S is the simulation time as computed by the causality-control protocol, which is based on the window-based synchronization protocol described in [11].

T_W, the scaled wall-clock time, is a linear function of the processor time taken from a reference processor, i.e., $T_W = Ptime \cdot K_{ratio}$, where P_{time} denotes the processor time and K_{ratio} is a control parameter that can be changed by the user.

Performing the simulation in the above described way allows us to synchronize the evolution of the simulation time with a linear function of the processor time. Our system is therefore capable of reproducing and visualizing the dynamics of the real system, with the time scaled by a factor. By changing K_{ratio}, we can control the speed at which the simulated system state evolves. For example, we can

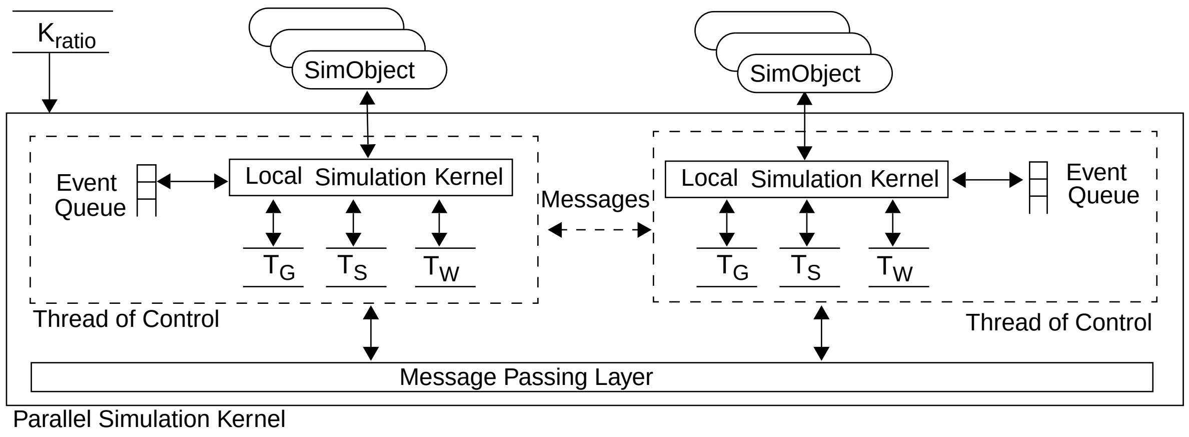


Figure 2: Realization of the Parallel Simulation Kernel.

slow down or "speed up" the simulation. (Speeding up the simulation is obviously constrained by the performance characteristics of the hardware.)

By processing an event only when its time-stamp is smaller than T_S -- i.e., when no causality violation can occur -- we follow the conservative approach to parallel simulation. We eschew the optimistic approach, which requires roll-back and state saving mechanisms, mainly because that approach makes it difficult to build simulation objects independent of the simulation kernel [12]. Our objective is to have a clean separation between the emulated system and the simulation kernel, rather than improving the performance of the simulation kernel. (It is argued that, in general, optimistic simulation allows for better performance of a parallel simulation kernel [7], although our application falls into a class where conservative simulation exhibits a comparable speedup [11].)

The parallel simulation kernel has been implemented on two platforms with different hardware architectures: the IBM SP2 (distributed memory) and the KSR-1 (shared memory). These implementations differ with respect to realizing message passing, object references, and clock synchronization. While message passing is the "natural" programming paradigm on the SP2, we had to implement this functionality on the KSR-1 and put much work into performance optimization. On the KSR-1, there is no distinction between local or remote objects, whereas on the SP2, all objects are local and a mapping from objects to processors is required. This difference is particularly important for realizing interactive monitoring and control capabilities. On the KSR-1, these functions can be easily implemented by having centralized entities reading and writing shared variables. On the SP2, a set of local monitoring and control objects is needed to perform the same task in a distributed way. As for clock synchronization, an arbitrary processor is chosen on the KSR-1 to provide the master clock. It continuously writes the value of its processor time into a shared variable which is visible to all other processors. The estimated time on a local simulation kernel is the value of the local cache copy of the master clock. These time variables are very well synchronized--usually within 1 µs--, since the KSR-1 has special hardware for maintaining cache consistency. On the SP2, a clock synchronization protocol is used by each local simulation kernel to estimate the processor time of a specific master processor, which allows for synchronization in the order of 1ms.

2.2 Emulation Support

Figure 3 shows a data-flow diagram of the software architecture for the support of real-time visualization and interactive control. Its design follows the monitoring-control paradigm. Monitoring is realized as a continuous activity, whereby a stream of data produced by the network emulator is consumed by the operator interface in the graphics workstation. Interactions among components involved in the monitoring process are asynchronous, via reading and writing shared objects, which allows them to run on different time scales. Control operations, on the other hand, are event-driven and are based on the client-server paradigm.There are two types of control operations: (1) *emulation control operations*, which alter the behavior of the emulated system and are executed by the network emulator, and (2) *monitoring control operations*, which affect the monitoring activity. *Emulation control operations* include changing the input load on the system and modifying the control policies with which the control mechanisms run. Monitoring control operations tune components in both the emulation system and the manager station. They regulate the part of the network state that can be visualized on the manager station, as well as the amount and the granularity of data that is carried over the network, by, for example, adjusting the sampling interval and the size of the monitoring window. Inside the manager station, the information collected from the various network objects can be abstracted, correlated and displayed on the screen in different ways. By varying the quantity and granularity of information collected and sent over the network, our system can be tailored to different computing and network platforms.

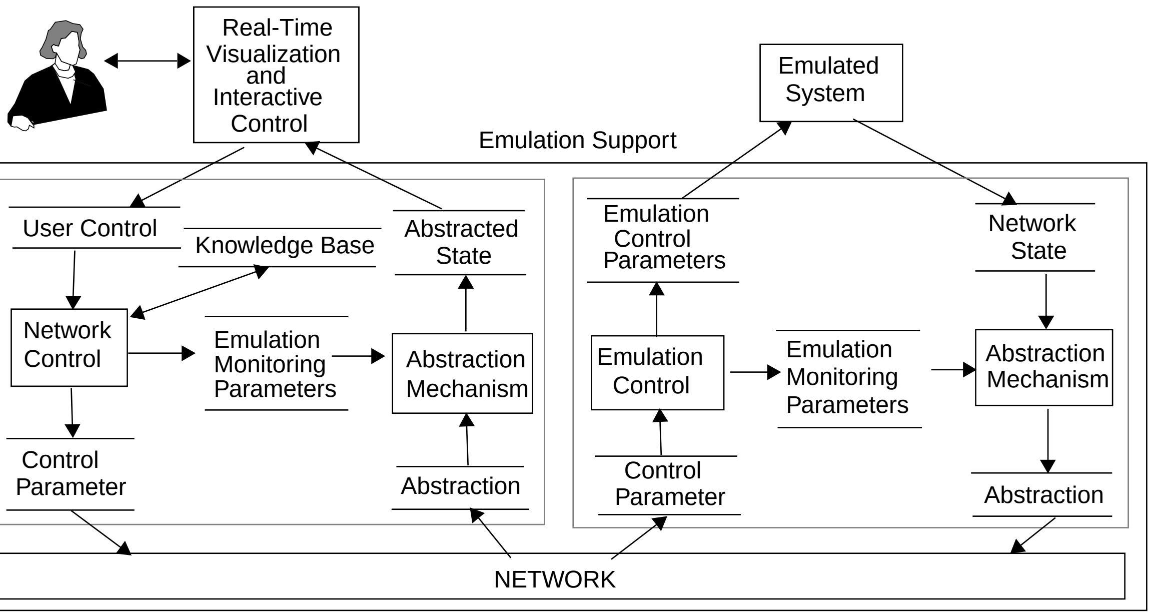


Figure 3: Software architecture for the Emulation Support module.

2.3 Real-Time Visualization and Interaction

The design of the Real-Time Visualization and Interactive Control module focuses on the selection capabilities, high-level control primitives available to the operator, and visual representation of network states. These modules allow an operator to try what-if scenarios, invoke specific network services during run-time, emulate network management operations, and visualize the effect of these operations.

Operations performed by the human operator through the interface are categorized into three groups. The first set relates to defining the input traffic characteristics for each network node. The second set enables a variety of monitoring functions. By selecting objects (switches, links, network regions, etc.) on the network map, together with the desired monitoring option, the operator can visualize, in real-time, various abstractions of the network state, including traffic intensities and network utilization. Another set of interface operations refers to changing management parameters, which allow operators to tune the behavior of mechanisms in the traffic control system.

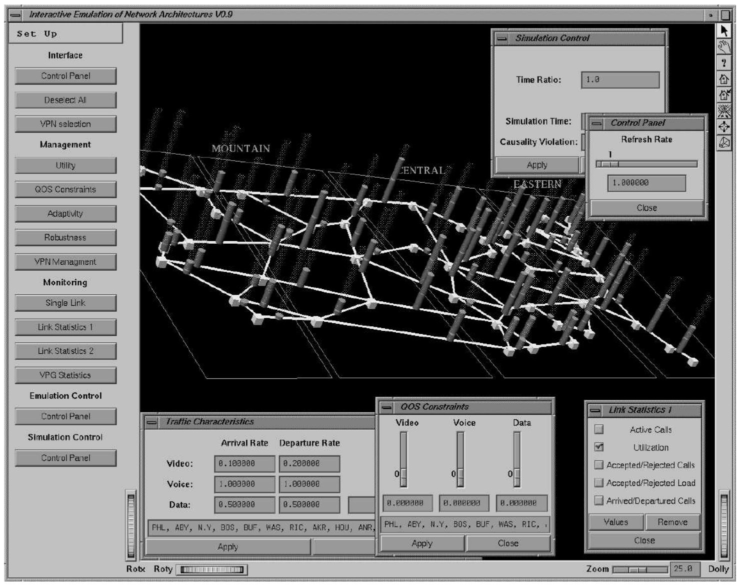
2.4 Emulated System

The design and implementation of the emulated system depends largely on the target architecture. The description of the modeling concepts used in the design of a performance management architecture is described in section 4.

Object interactions are modeled as asynchronous message passing. For example, a connection setup in an ATM network is modeled as an exchange of messages between the connection managers and link admission controllers. Logically, a mail-box is associated with each object. If object A wants to send a message to object B, it invokes a send function that inserts the message into B's mail-box. Conversely, object B receives messages from other objects invoking a receive function that removes them from its mail-box. The receive and send functions are realized as the `exec` and `sendMessage` functions, respectively.

3. Realization of the Platform

In our implementation, both the emulated system and the simulation kernel (coded in C++) run on a supercomputer (either the KSR-1 or SP2) located at the Cornell Theory Center (CTC) in Ithaca, New York. The real-time visualization and interactive control module resides on an SGI Indigo2 workstation at Columbia University. It is written using Open Inventor, a 3D graphics tool kit based on Open GL, and runs as a UNIX process that communicates with the emulation support module via *`UNIX System V`* shared memory. The emulation support module is distributed on the two machines. These machines communicate through NYNET, an ATM network that connects several research laboratories in New York State. In order to emulate a network of a certain size in real-time, a large number of operations must be performed per second. For example, in one of our scenarios, approximately 13,000 operations per second are executed for a 50 node network, in response to about 1000 connection requests per second. The required performance is achieved using a massively parallel machine. We have implemented the emulation platform on a 128 node KSR-1 and a 512 node IBM SP2, both of which are located at the Cornell Theory Center. The KSR-1 is a shared memory parallel machine with hardware support for maintaining cache consistency. Each node has a 40 MFlops processor and 32 Mbytes of local cache memory. Accessing memory not available on the local cache takes between 8.75 ms to 30 ms, depending on the distance between



Time Ratio: 1.0
Simulation Time:
Causality Violation:
Refresh Rate: 1 — 1.000000

	Arrival Rate	Departure Rate
Video:	0.100000	0.200000
Voice:	1.000000	1.000000
Data:	0.500000	0.500000

PHL, ABY, N.Y, BOS, BUF, WAS, RIC, AKR, HOU, ANR,

Video	Voice	Data
0.000000	0.000000	0.000000

PHL, ABY, N.Y, BOS, BUF, WAS, RIC,

Zoom 25.0

Figure 4: Operator interface for managing a multimedia network.

nodes. The operating system is MACH, and we use the OSF-1 pthread package to exploit the shared memory architecture of the machine. The IBM SP2 has a distributed memory architecture. Each node consists of a 266 MFlops POWER2 processor, has between 64 and 2048 Mbytes of memory, and runs its own copy of the AIX operating system. The parallel programming environment we use is based on the Message Passing Interface (MPI) package. For the current MPI implementation on the SP2, sending a 200 byte message from one node to another takes about 50 ms.

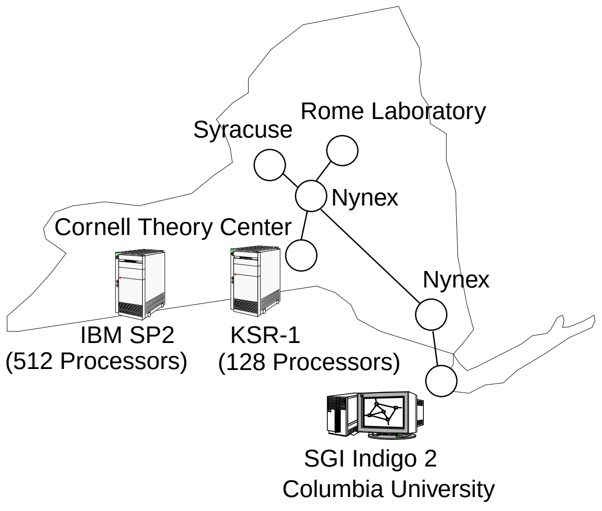


Figure 5: Hardware configuration of the interactive emulation platform.

The machines at the Cornell Theory Center and at Columbia University are connected by a permanent virtual circuit connection (PVC) on NYNET. The bandwidth B required by the connection is given by B = LM * NO * R, where LM is the size of the monitoring data unit, NO is the number of objects being monitored and R is the refresh frequency. When NO = 500 objects are being monitored (which is the case in one of our scenarios) and the data is refreshed R=1 times per second, the bandwidth needed is approximately 2 Mbit/s, given the size of the monitoring data unit LM=512 bytes. When the communication between the two machines is performed over Internet, the number of objects monitored and the refresh frequency must be reduced to accommodate the lower bandwidth connection. In this case, our experience shows that, even with a reduced monitoring load, the packets carried over the Internet connection can experience delay variations of up to a few seconds. The ATM connection currently in use has been shown to be very useful for our purposes, providing higher throughput with lower delay jitter.

4. Application

In this section we illustrate how the emulation platform, described in the previous sections, can be used to prototype control architectures for high-speed multimedia networks. Using our platform, we are able to study the behavior of control systems for large networks without first building a prohibitively expensive testbed. In particular we are able to study the behavior of the prototype system under different load patterns, network configurations and size. We are able to monitor and collect information concerning any element of the prototype system by simply reading the states of controllers --- a task that is implemented as a generic functionality of the emulation control system. Our platform allows system designers to shorten the design/implementation/test cycle, by providing an environment that permits adding and/or modifying subsystem functionality (e.g., changing the routing algorithm or introducing a multi-cast service) in a rapid way.

The control and management systems of a high-speed multimedia network are composed of a large number of controllers (such as connection managers, routers, call admission controllers, etc.) which interact with each other and run on different time-scales. The task of the control system is to provide end-to-end quality-of-service (QOS) to network services while utilizing network resources in an efficient way. The traffic control system operates under specific assumptions on the traffic statistics, the behavior of applications, and QOS demands. However, given the dynamic nature of multimedia traffic, the conditions under which a multimedia network actually operates may be different from those considered during the design phase. Therefore, a management system capable of taking control actions to dynamically adjust the network behavior is required. In this way, the goal of efficiently providing services with QOS guarantees to the user applications can be met, under varying load patterns, changes in user behavior, and other unpredictable conditions.

We have developed a management and control architecture for multimedia network services, which focuses on supporting human operator with high-level controls and dynamic visualization abstractions in pursuing their objectives of service effectiveness and efficiency (e.g., guaranteeing QOS to different traffic classes and obtaining a high degree of multiplexing) [13, 5].

In order to validate and refine our approach, we built a prototype of this architecture on the emulation platform. The emulated system runs the components of the traffic control system and management agents, emulating the behavior of routers, connection managers, link admission controllers, etc. Each controller is implemented as one or more objects in the emulated system module. The simulation kernel implements the communication mechanism for exchanging information among the controllers, and it is used to model the communication and computational delays of the system to be emulated. The interactive interface has two purposes: (1) to control the emulation (execution speed, network load, etc.) and (2) to provide the functions of a network operator interface, i.e., different views and control capabilities (see Fig. 4). The emulation platform allows us to execute management operations for a variety of network configurations and load scenarios, and to observe their effect on the evolution of the network state in real-time. (The network state is the aggregation of the states of the controllers and statistical abstractions thereof.) In an interactive way, we can study what would happen, if an operator performs a specific action in response to a critical situation, such as network overload or link failure.

A configuration we often use for experiments is that of a 50 node cross-country broadband network. Its control and management system contains some 200 controllers, which run asynchronously and on a variety of time-scales. In our architecture, resource controllers, e.g., link admission controllers, are composed of four independent mechanisms [13], each of which is implemented as a simulation object. The network control system can handle a load of some 20 calls per second and per node when the emulation runs on a 64 node KSR-1 processor set.

We performed a number of experiments to study the influence of management parameters, in particular QOS parameters and parameters related to the responsiveness of the control system [5]. We demonstrate how a network operator can influence the network state using high-level controls. For example, we show how the traffic mix in a multimedia network can be influenced by changing the blocking constraints of network services, allowing an operator to favor one class of traffic, such as audio, over another class, such as video. The effect of such an operation can be observed at our operator interface--the network state moves from one operating region into another. A demonstration of the management and control architecture running on our platform was given at the last ACM Multimedia conference [6].

References

[1] A. Dupuy, J. Schwartz, Y. Yemini, D. Bacon, “NEST: A Network Simulation and Prototyping Testbed,” *CACM*, vol. 33, no. 10, pp 63-74, Oct 1990

[2] T.D. Blachard, T.W. Lake, “Distributed Simulation with Locality,” in *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, pp. 195-198, June1995

[3] Arthur Chai and Sumit Ghosh, “Modeling and distributed simulation of a broadband-ISDN network,” *IEEE Computer*, vol. 26, pp. 37–52, September 1993.

[4] M.C. Chan, H. Hadama and R. Stadler, “An architecture for broadband virtual networks under customer control,” IEEE Network Operations and Management Symposium, (Kyoto, Japan), April 1996.

[5] M.C. Chan, G. Pacifici, and R. Stadler, “Managing real-time services in multimedia networks using dynamic visualization and high-level controls,” in *Proceedings of the ACM Multimedia*, San Francisco, CA, November 1995

[6] M.C. Chan, G. Pacifici, and R. Stadler, “Real-time emulation and visualization of large multimedia networks,” in *Proceedings of the ACM Multimedia*, Demonstrations Program, San Francisco, CA, November 1995

[7] R. Fujimoto, “Parallel discrete event simulation,” *CACM*, October 1990, Vol. 33, No. 10, pp. 31-53

[8] R.C. Hofer, M.L. Loper, “DIS Today,” in *Proceedings of the IEEE*, Vol. 83, No. 8, pp. 1124-1137, August 1995

[9] S. Kheradpir, W. Stinson, R. Chipalkatti, and G. Bossert, “Managing the network manager,” *IEEE Communications*, vol. 30, pp. 12–21, July 1992

[10] A. M. Law and M. McComas, “Simulation software for communications networks, the state of the art,” *IEEE Communications Magazine*, vol. 32, pp. 44–50, March 1994

[11] D.M. Nicol, “The cost of conservative synchronization in parallel discrete-event simulations,” *Journal of ACM*, 40(2):304-333, April 1993

[12] D. Nicol, P. Heidelberger, “On extending parallelism to serial simulator,” in *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, pp. 60-67, June1995

[13] G. Pacifici and R. Stadler, “An architecture for performance management of multimedia networks,” in *IFIP/IEEE International Symposium on Integrated Network Management*, (Santa Barbara, California), May 1995

[14] J.S. Steinman, “SPEEDES: Synchronous parallel environment for emulation and discrete event simulation,” *Advances in Parallel and Distributed Simulation*, vol. 23, pp 95-103, Jan 1991

[15] J.S. Steinman, C.A. Lee, L.F. Wilson, D.M. Nicol, “Global virtual time and distributed synchronization,” in *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, pp. 139-148, June1995

[16] B.W. Unger, D.J. Goetz, and S.W. Maryka, “Simulation of SS7 common channel signaling,” *IEEE Communications Magazine*, vol. 32, pp. 52–62, March 1994