INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. 2002; **15**:713–739 (DOI: 10.1002/dac.560)

A batch-type time-true ATM-network simulator—design for parallel processing

Michael Logothetis*,†,‡,[1] and Fotios Liotopoulos[2,§]

[1] *Wire Communications Laboratory, Department of Electrical & Computer Engineering, University of Patras, 26500 Patras, Greece*
[2] *Computer Technology Institute, Akteou 11 & Poulopoulou, Thesseo, 11851 Athens, Greece*

SUMMARY

This paper presents a new type of network simulator for simulating the call-level operations of telecom networks and especially ATM networks. The simulator is a pure time-true type as opposed to a call-by-call type simulator. It is also characterized as a batch-type simulator. The entire simulation duration is divided into short time intervals of equal duration, t. During t, a batch processing of call origination or termination events is executed and the time-points of these events are sorted. The number of sorting executions is drastically reduced compared to a call-by-call simulator, resulting in considerable timesaving. The proposed data structures of the simulator can be implemented by a general-purpose programming language and are well fitted to parallel processing techniques for implementation on parallel computers, for further savings of execution time. We have first implemented the simulator in a sequential computer and then we have applied parallelization techniques to achieve its implementation on a parallel computer. In order to simplify the parallelization procedure, we dissociate the core simulation from the built-in call-level functions (e.g. bandwidth control or dynamic routing) of the network. The key point for a parallel implementation is to organize data by virtual paths (VPs) and distribute them among processors, which all execute the same set of instructions on this data. The performance of the proposed batch-type, time-true, ATM-network simulator is compared with that of a call-by-call simulator to reveal its superiority in terms of sequential execution time (when both simulators run on conventional computers). Finally, a measure of the accuracy of the simulation results is given. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: simulation; VC routing networks; ATM networks; call-level traffic; parallel processing; bandwidth control

*Correspondence to: M. D. Logothetis, Wire Communications Laboratory, Department of Electrical and Computer Engineering, University of Patras, 26500 Patras, Greece.
† E-mail: m-logo@wcl.ee.upatras.gr
‡ URL: http://www.wcl.ee.upatras.gr/m-logo
§ E-mail: liotop@cti.gr

Contract/grant sponsor: University of Patras/Greece

Received February 2001
Revised February 2002
Accepted May 2002

1. INTRODUCTION

The wide application of simulation in telecommunications has given rise to several simulation-specific computer-languages (e.g. SIMSCRIPT, GPSS), as well as to software packages for the simulation of communication networks (e.g. COMNET, OPNET). However, we believe that in many cases software packages cannot always generate the specific network conditions that the researchers may wish to study. Even if some of these packages offer the users the capability of extending and customizing the source code according to their specific needs, this is often cumbersome and expensive. On the other hand, most of the times it is not so difficult for a researcher to build a custom-made network simulator in order to completely control its computer implementation and, consequently, the simulation conditions. Furthermore, we believe that general purpose programming languages (GPPL), used as simulation languages, are more flexible than actual simulation languages and in most cases they can achieve faster execution times [1]. Therefore, in this paper we propose a simple and fast method for simulating the call-level operations of telecom networks and especially of ATM networks [2], by using a GPPL. For a researcher who wishes to build his/her own network simulator, the use of his/her familiar GPPL, an easy-to-implement data structure and a fast-execution simulation method are key points.

The simulation of telecom networks, conveying traffic end-to-end through communication links, is a discrete-event simulation process [3]. The number of active calls (or connections) of the subscribed network services changes by means of the discrete events of *call-arrival* and *call-termination*. Discrete-event simulation can be modelled by three approaches:

i. *Event-oriented modelling*, where the simulation model is described as a sequence of call-arrival and call-termination events.
ii. *Process-oriented modelling*, where the simulation model is described by the behaviour of the call in the network.
iii. *Activity-oriented modelling*, where the simulation model is described by the initiation and termination of activities, which depend on the state of the network.

Both the event-oriented modelling and the activity-oriented modelling are suitable for GPPL. In simulating telecom networks there is no special need for activity-oriented modelling, since the service-time of a call is independent from the number of existing calls in the network or the number of busy trunks. Our simulation approach is based on the event-oriented modelling and realizes the *time tracing* (*time-true*) simulation method [4]. Our main objective is to reduce the execution time of the simulator. The execution (CPU) time is an important factor in simulating large-scale systems. For example, for the study of the performance of a telecom network with complex routing or bandwidth control, the operation of the network must be simulated for several hours or even days; this implies many hours of execution time. To alleviate this problem, we introduce a new technique for simulation, the *batch processing* technique.

A technique that is widely applied in the simulation of telecom networks is to advance the simulation according to arrival events, on a call-by-call basis (call-by-call simulation) [1,5]. Contrary to this approach, we propose to advance the simulation not on a call-by-call basis, but instead, once for every a large number of arrival calls (i.e. a call-batch), introducing the batch-type time-true technique (batch-type simulation) [6]. The entire simulation duration is divided into short time intervals of equal duration, t. During t, a batch processing of call origination and

termination events is executed and the time-points of these events are sorted. The number of sorting executions is drastically reduced compared to a call-by-call simulator, resulting in considerable timesaving. Furthermore, in an attempt to reduce the execution time as much as possible, we have designed the data structures of the simulator so that they are well fitted to parallel processing techniques, in order to be possible for the simulator to be implemented on a parallel computer.

Having implemented the simulator in conventional, sequential computers, we subsequently applied a 'parallelization' process on the software in order to achieve its parallel-computer implementation; this is a current approach for parallel processing [7]. By the term parallelization we mean to adapt an algorithm or software for running on several threads of execution (either processes or processors). To simplify the software complexity of parallel processing, we dissociate the core network simulation from other installed functions (such as the virtual path bandwidth control) and we apply the parallelization process only on the core network simulation. The parallelizable design of the data structure of the simulator consists of organizing data by virtual paths (i.e. end-to-end links) and then distributing them among processors, which all execute the same set of instructions on this data.

This paper is organized as follows. Section 2 gives an overview of the key concepts of the ATM network technology and the call-level operations of ATM-networks that are involved in the simulation, that is: (1) virtual path, (2) network topology, (3) service-classes, (4) call-level operations (such as, virtual path bandwidth control, bandwidth rearrangement time and virtual circuit routing control). Also, Section 2 lists various options and built-in functions of the simulator. In Section 3, we present the batch-type, time-true simulation method and elaborate on the motivation behind this new method. Furthermore, we present a heuristic procedure, whereby we determine the small time-interval t of the batch-type simulation method, depending on the computer system that we use. In Section 4, we describe the proposed data structures of the simulator that is suitable for parallel processing. In Section 5, we explain the parallelization procedure of the batch-type time-true (BTT) ATM-network simulator. First, we dissociate the core simulator from its built-in functions and describe the parallelization process applied to the sequential version of the simulator. We present the possible methods for parallelization that are suitable for the data structures of the simulator and, finally, we present our choice for its parallel implementation. Section 6, compares the performance of our simulator against that of a call-by-call simulator, with respect to the execution time and the required computer memory, when both simulators run on conventional computers. Also, a measure of the resulting accuracy is given for the BTT simulator, when either a whole network or a single end-to-end link is considered. As a conclusion, Section 7 summarizes the main points of the proposed batch-type, time-true, ATM network simulator.

2. PRELIMINARIES—SIMULATOR OPTIONS AND FUNCTIONS

We aim at simulating the call-level operations of telecom networks. By the term telecom networks we mean connection-oriented communication networks, or virtual circuit (VC) routing networks, according to the terminology of the network layer (3rd layer) of the OSI model, where the dominant function of the layer service model is the function of call set-up. Datagram based networks (like IP networks), where no call set-up function exists, are for further study.

In this section, we describe the considered architecture and call-level operations of ATM networks (VC based networks) that are involved in the simulator we have implemented.

2.1. Virtual path

A virtual path (VP) is a logical link between two nodes (endpoints) in ATM networks [8]. It consists of a bunch of virtual channels (VC). A VC is a unidirectional communication capability for the transport of ATM-cells and, from the bandwidth management viewpoint it consists of bandwidth allocation units, also known as trunks. Therefore, a VP is a set of trunks between two nodes and it is an abstraction of a physical path assembled as a set of physical transmission links. For each pair of nodes several VPs may exist, routed through different physical paths. VPs are the only way to access the underlying network from the nodes. It is up to the implementation of the network to convey the traffic for a VP by using an appropriate route. The actual configuration and routing structures are transparent for the agents using the ATM network [9]. When we consider STM networks for simulation (instead of ATM networks), then a VP is equivalent to the allocated bandwidth of a unidirectional end-to-end link of the STM network.

2.2. Network topology

For network reliability purposes, it is often a good practice to have at least two different engineering paths between two nodes, routed via different transmission links. If this happens, some paths (routes) may serve as auxiliary pathways in case of a link failure, or for load balancing within the network. Therefore, the considered network is based on the ring topology, which is one of the favourite topologies of high-speed networks, such as ATM networks.

Figure 1 shows an example of a simple ring topology of an ATM network with 9 (terminal) nodes. It shows that between nodes 1 and 4 we can implement two VPs on different routes. Nodes (or ATM switches) are the apparent part of the network, whereas the cross-connect systems (invisible pairs of switches) bind the physical links to form the communication frame or 'backbone network'. Through the cross-connect systems, we can join separate rings to form the

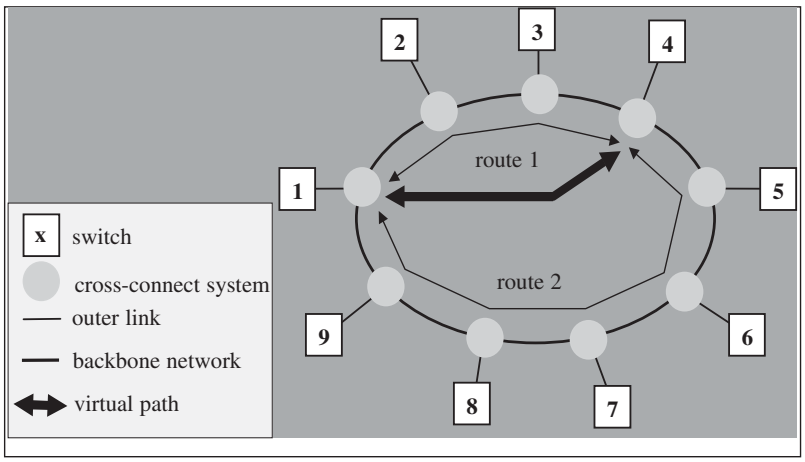


Figure 1. ATM network topology.

topology of a larger network. In this way, we can consider any network topology as composed of ring sub-networks.

When we deal with medium or large-scale networks, computer-aided description of network topology is needed. For example in a 10-node network, the routes of 90 paths (VP) (often called routing table) must be defined, considering only the shortest path for each origin–destination pair. When developing a network simulator, additional effort is required for developing software modules to handle input data (such as the network topology and routing tables) compared to using a commercial simulator.

2.3. Service-classes

The ATM network design also includes features about service-classes as suggested in the specifications of the broadband integrated services digital network (B-ISDN). The idea behind is that, as communications are becoming more and more important, different types of data are transmitted with very diverse requirements, either in bandwidth or quality of service (QoS) per call. Such data types include voice, video, file transfers, raw data, remote control information, etc. In designing the simulator, we assume that a VP may accommodate several different service-classes.

From the simulation point of view, the different service-classes, at the call set-up phase, are distinguished in two main categories: (a) service-classes with non-elastic traffic and (b) service-classes with elastic traffic [10]. QoS guarantee service-classes are considered of non-elastic traffic, which can be modelled as circuit emulated traffic through the concept of equivalent bandwidth. However, the concept of equivalent bandwidth is not directly applicable to elastic services (like ABR—available bit rate service). Nevertheless, it is still possible to model elastic and non-elastic traffic in a common way, since there is an amount that is constant for both service categories; this is the 'total' offered traffic-load [1]. The concept of time is inherently included in the traffic-load, as traffic-load is the product of arrival rate by the holding time. Total offered traffic-load is the product of traffic load by the required bandwidth per call. Or, by incorporating the arrival rate to the required bandwidth per call, we could assume the total offered traffic-load as the product of the holding time by the required bandwidth per call. Therefore, at the stage of call set-up, an elastic service-class can be considered a concatenation of non-elastic service-classes [10]. For example, suppose an elastic service-class with maximum bandwidth requirements per call of 2 Mbps and minimum bandwidth requirements per call of 1 Mbps. We assume that such a call arrive to the network and initially asks the maximum bandwidth of 2 Mbps while requires a holding time of 150 s. Since it belongs to an elastic service-class the network may start serving the call with 1.5 Mbps, which means that the anticipated holding time becomes 200 s (since $2*150 = 1.5*200$). For each call of an elastic service-class, additional sorting execution is required in the simulation, if the call is served with other than the initial (maximum) bandwidth requirement.

2.4. Call-level operations

In ATM networks, network/traffic control can take place at two distinct levels, the *call-level* and the *cell-level*, which correspond to the distinction of traffic in call and cell components, respectively. The cell-level studies are usually restricted either to a single link, or to a single switch, or to a single switching node-pair [3]. This is because either they take into account detailed traffic parameters and thus the information processing becomes extremely large if we

cope with the whole network, or, because, there is no need to extend the study to a whole network, since most of the cell-level traffic-control functions are applied to a network component independently from the others. Fortunately traffic/network control functions can be applied only to the call, or only to the cell level. Only the function of connection admission control (CAC), which is basically a call-level function, requires feedback from the cell-level.

In the call-level network management, the performance is evaluated by the call blocking probability (CBP). In simulation, the CBP is determined simply by

$$CBP = \frac{\text{Number of calls lost}}{\text{Number of calls offered}}$$

In our study, we have concentrated in simulating those call-level operations, which drastically influence the global performance of network, under constraints posed by the bandwidth capacities of transmission links [11]. These operations are virtual path bandwidth control and virtual circuit control. Any CAC functions will be applied by ignoring the cell-level.

2.4.1. Virtual path bandwidth control (*VPB*). Virtual path bandwidth (VPB) control concerns the bandwidth distribution to the network's VPs [12–14]. It changes the bandwidth usage of VPs according to the traffic fluctuations in order to improve the network performance. Using traffic measurement, VPB control follows the traffic variations and assigns to some VPs more or less bandwidth, accordingly, so that to minimize the maximum CBP among all the switching node-pairs. VPB control usually incorporates trunk/bandwidth reservation control, when circuit emulated traffic is considered. The network reserves some fraction of the free bandwidth (a certain number of trunks) of a commonly shared VP among service-classes. The reserved number of trunks is only available to the service-class, which requires more bandwidth per call (e.g. high-speed calls). The purpose is to maintain the desired grade-of-service for each service-class.

2.4.2. Bandwidth rearrangement time (*BRT*). VPB control results in bandwidth rearrangement. An important function of the simulator is the measurement of the bandwidth rearrangement time (BRT) [15]. Bandwidth rearrangement cannot be completed at once, because of the already existing call-connections in the VPs (end-to-end links) whose bandwidth capacity must be reduced [16]. These connections are released on a call-by-call basis. During this time, the network is in a special state: the bandwidth released by these terminated calls is set in an idle state and cannot be used by newly arriving calls. When all pairs of nodes that have to reduce their bandwidth have put it to the level imposed by the VPB controller, then the bandwidth changes are enabled. Subsequently, the network operates normally again, with the new bandwidth distribution over the VPs. The time spent waiting for the call-connections to be released, from the time-point when the bandwidth rearrangement begins, until its realization is complete, is called bandwidth rearrangement time (BRT).

2.4.3. Virtual circuit routing control (*VCR*). This control is also known as dynamic routing and competes with VPB control in efficiency [17]. The least loaded route is selected to convey every arriving call. VCR control is particularly efficient when the traffic fluctuation rate is small.

In view of the above, we have developed a simulator to simulate the call-level operations of ATM networks. Herein, we give a short description of the most important options and functions of the simulator.

Options

- Automatic, or not, determination of the stabilization (relaxation) time of the simulation.
- Homogeneous, or non-homogeneous, Poisson process for arrival calls.
- Contiguous, or non-contiguous, bandwidth occupation by the calls.
- Calls may or may not change their bandwidth requirements at call set-up.
- Complete sharing policy of a virtual path, or with bandwidth reservation scheme among the service-classes of the network.
- Calls can, or cannot be accommodated partially by two virtual paths.

Functions

- Measurements of carried traffic, blocking probabilities, busy trunks, etc. [18].
- Evaluation of virtual path bandwidth control schemes [18].
- Estimation of bandwidth rearrangement time [15,18].
- Evaluation of dynamic routing schemes [17].
- Evaluation of retry and threshold models used as call admission control models [10,19].

3. MOTIVATION FOR THE BATCH-TYPE TIME-TRUE ATM-NETWORK SIMULATOR

Having adopted event-oriented modelling as suitable for the simulation of telecom networks, we have to decide on which of the two subsequent methods used by the event-oriented modelling we shall follow: the *roulette method* or the *time-true method* (see Figure 2) [1,4].

In the 'roulette method', which is also called 'the Markov-chain method', a Markov chain representing changes of the system states is considered, while the concept 'time' disappears [4]. However, the concept of time is one of the most important features of network simulators. Many times we wish to assess the transient behaviour or the performance of a network within a certain time period. Thus, the time-true simulation method is the most promising simulation method, because it traces the time points at which events occur, and, therefore, we follow it in the proposed simulator. This method may be unavoidable for performance evaluation of telecom systems that involve complex scheduling strategies, or when arbitrary distribution of

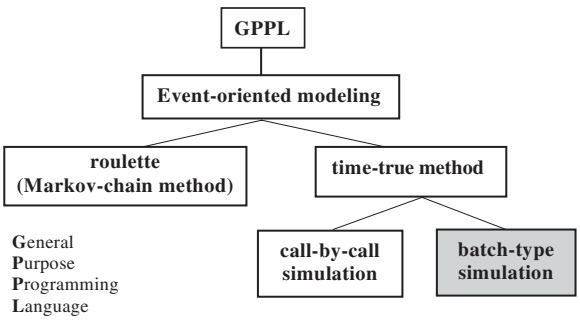


Figure 2. Simulation methods for event-oriented modelling implemented by GPPL.

service time is considered. It is also worth mentioning that it is possible to combine the two methods. The time-true method can be applied in the part of a simulated process where time measurements are necessary, while in the rest part of the process the roulette method is applied. Nevertheless, we do not adopt the latter in order to produce a simple simulation method.

To realize the time-true method, general-purpose programming languages are appropriate [1]. Figure 2 shows that a GPPL is suitable for event-oriented modelling according to the time-true method. It also shows that two simulation techniques exist, whereby we can realize the time-true method: the call-by-call simulation (old method [1,4]) and the batch-type simulation (proposed new method [6]). The conventional method of call-by-call simulation is to advance the simulation according to the arrival events, call by call (call-by-call simulation). Instead, we propose to advance the simulation not call by call, but for every large number of arrival calls (batch), introducing the batch-type, time-true technique (batch-type simulation).

According to the time tracing simulation method of event-oriented modelling, the ATM network is modelled by describing state changes by events such as call origination and termination. The time points of the events are stored in a timetable (e.g. a time array). In simulators of call-by-call type, the simulation proceeds to the time-point of the earliest event. Each time a call origination occurs and the network service (free trunks) is available at that time (otherwise the call is lost or delayed), the call termination time point is determined and stored in the timetable. The table is sorted so that the next earliest event is found (search for a minimum). Figure 3 illustrates the principle of the time-true method. On the left-hand side, it gives a

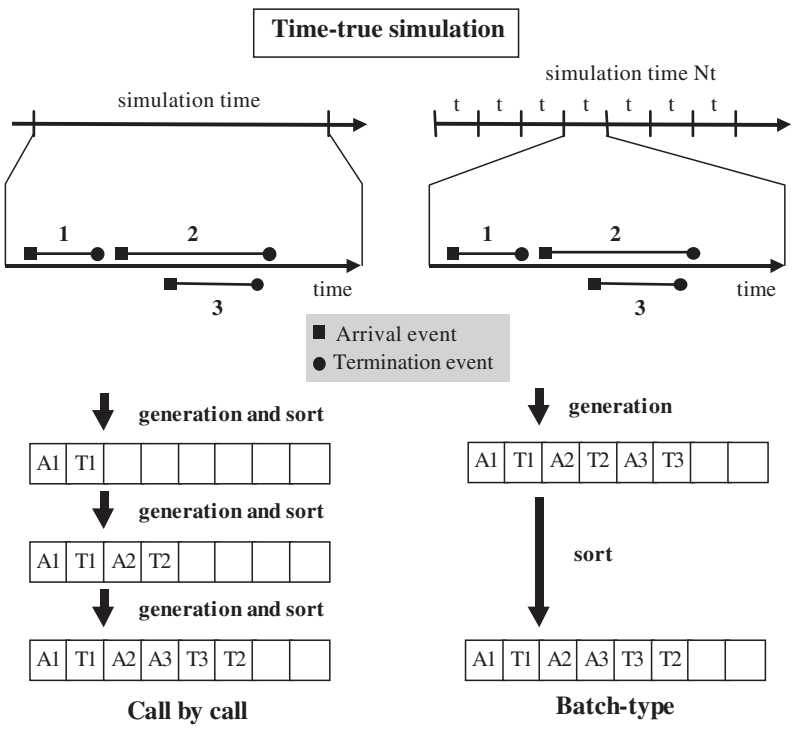


Figure 3. Methods for time-true simulation.

simulation example of the first three arrival calls (1, 2 and 3) in an end-to-end link of a telecom network. The arrival time-point (■) of the first call is Al, whereas its termination time-point (●) is Tl. These two numbers are stored in the time array. For the second call, the arrival and termination time points are A2 and T2, respectively, which are also stored in the time array. Then, the time array is sorted. Likewise, for the third call, A3 and T3 are the call arrival and termination time points, respectively. These time points have to be inserted in the time array between A2 and T2 in order for the time array to be sorted. This procedure will continue with the next calls until the end of the predefined simulation interval. The result from this call-by-call simulation is that there are as many sorting executions as the number of served calls. Therefore, the call-by-call simulation is time consuming. Of course, the required sorting can be performed by a simple binary insertion (binary search), in order for the new event to be inserted in the correct place within the time array. Nevertheless, since this is repeated for each event and due to the great number of events when network (not link) simulation is considered, the procedure of keeping the time array sorted is very time consuming.

In order to save considerable computer-time, we propose a kind of batch processing of the events. This is achieved mainly by reducing the sorting executions. The entire simulation duration is divided into N short time-intervals, each of equal duration t. During each small interval t, all calls are processed, but only one sorting operation is executed. Therefore, the proposed simulation technique can be characterized as a batch-type simulation. Sorting is performed N times during the entire duration of the simulation. As an example, on the right side of Figure 3, we show three calls (1, 2 and 3) that are generated during one small time-interval t. The arrival and termination time-points of these calls are generated sequentially and stored in the time array, which is sorted once during the interval t. Although, some of the arrival calls will not be served, the termination time-points of all calls are determined, as if they were going to be served.

3.1. Determination of the small interval t

The small interval t is a key factor for the efficiency of the simulator. By selecting t such as to cause the size of the resulting time array to be reasonable and consequently fast-sorted, considerable savings in CPU-time are achieved. More precisely, t depends on the total traffic volume and the scale of the ATM network and is defined through a heuristic optimization procedure, which is performed off-line.

First, we estimate the expected number of arrivals within the total scheduled time period T, for each service-class. The expected number of arrivals of service-class c_k is calculated as

$$c_k = \lambda_k T = (a_k/h_k)T$$

where, λ_k denotes the arrival rate of service-class k, and a_k, h_k denote the offered traffic load and the holding (service) time of calls, respectively, for service-class k.

We assume that the expected number of departures equals to the total number of arrivals. If K service-classes exist and n is the number of VPs in the network, the total length, L, of the time array (see Section 5, Figure 5) is estimated by

$$L = n \sum_{k=1}^{K} 2c_k$$

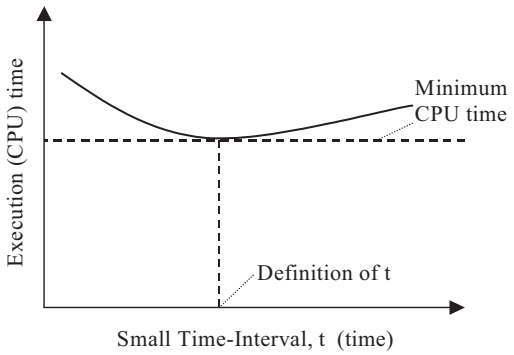


Figure 4. Definition of small time-interval, t, for the BTT simulator.

Then, we split L into N parts ($N = T/t$) and l is the length of the time array (i.e. $l = L/N$ about), which corresponds to the period of the small time-interval t. Eventually, the small interval t is defined by the minimum required CPU-time for sorting N times the time array of length l. A graph such as of Figure 4 is anticipated (compare with that of Figure 15). This is, because, if t is small then l is small, but N is large and, therefore, the CPU-time for sorting the time array of length l, N times, is large; on the other hand, if t is large, although N is small, l becomes large and therefore the CPU-time for sorting is also large. Obviously, the curve of Figure 4 is strongly dependent on the computer system and the sorting algorithm used. It is worth mentioning that a sorting execution is performed once for every large number of arrival calls in the batch-type simulator (e.g. one sorting execution for 50 000 arrivals), instead of one sorting execution for every 'served' call in call-by-call simulator. Besides, the length, l, depends on the period of pseudorandom numbers that are generated in the computer. This period should be greater than l (this is usually valid in modern software).

Considering a medium size network (i.e. of 10 terminal nodes) with realistic traffic loads, and an entire simulation duration, T, of 8–10 h, typical values for L is in the order of 10^6 for N about 100, l is in the order of 10^4 and t about 6 min.

4. DATA STRUCTURES

The data structures of the batch-type, time-true simulator basically consists of four arrays of equal length. Considering an ATM network with n VPs $(1, 2, \ldots, n)$ and each VP can accommodate up to k service-classes $(sc_1, sc_2, \ldots, sc_k)$, the four arrays are:

(a) The time array T (array of real numbers) contains the time-points of the call arrival or call termination events, A and T, respectively.
(b) The integer array VP contains the number of VP, which accommodates the call.
(c) The integer array E contains the kind of the event. The following coding is used: A positive number denotes an arrival event, while a negative number denotes a termination event. The absolute value of this number denotes the service-class of the call.
(d) A permutation array (integer array) produced after sorting the time array T.

More precisely, the data are generated and stored in the aforementioned arrays in the following order.

1. The time-points of all call arrivals during the small interval t, for VP_1 and service-class 1 (sc_1) are generated and stored in the time array T. The corresponding elements of VP and E array take the value of 1.
2. For each call arrival the termination time point is determined and stored in the time array T, after the end of all arrival time-points of the service-class 1, for VP_1. The corresponding elements in the *VP* array take again the value of 1, while the corresponding elements in the E array take the value of -1 (terminations of sc_1).
3. The time points of all call arrival during t for VP_1 and sc_2 are generated and stored in the time array T, after the end of all termination time-points of the service-class 1. The corresponding elements in the *VP* array take again the value of 1, while the corresponding elements in the E array take the value of 2 (arrival of sc_2).
4. For each call arrival of sc_2 the termination time-point is determined and stored in the time array T, after the end of all arrival time points of service-class 2 for VP_1. The corresponding elements in the *VP* array take again the value of 1, while the corresponding elements in the E array take the value of -2 (terminations of sc_2). In this way, the arrays are filled in for all service-classes (k) accommodated in VP_1. Then, the arrays are filled in for VP_2, starting from sc_1 to sc_k and then for the VP_3 and so on, until the last VP, VP_n. In one small interval t, the arrays are filled in, as Figure 5 illustrates. Then, the time array is sorted. Since we need to keep track of the correspondence between the sorted time array and the others, the permutations of the sorted-array T are kept in an additional array, the *permutation array*. Based on this, we know in which VP and service-class each element of the time array T refers to and whether it is an arrival or termination event. Figure 6 depicts the data structures before and after sorting, and how the permutation array serves as link between the others arrays (see also Figure 3).

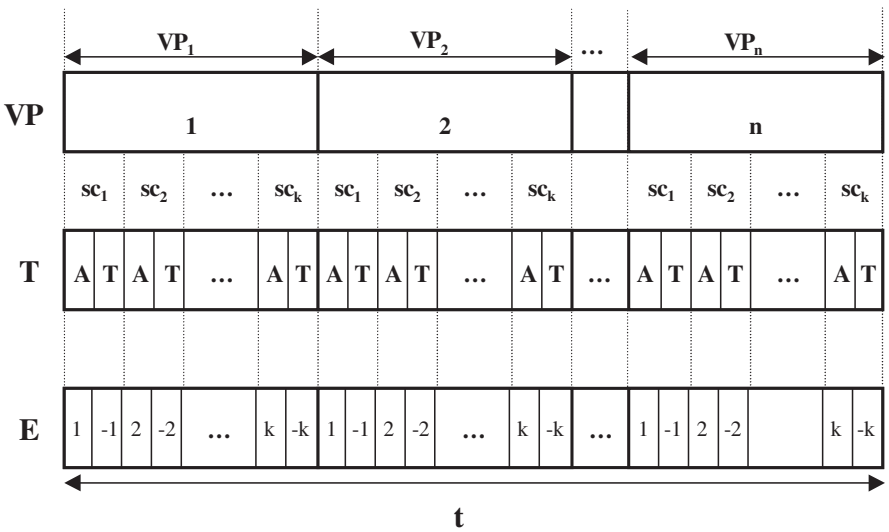


Figure 5. Data structures of the batch-type time-true ATM-network simulator.

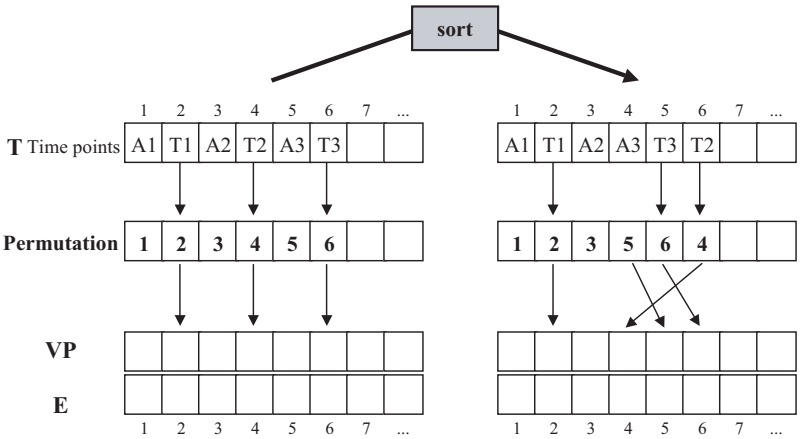


Figure 6. The role of permutation array.

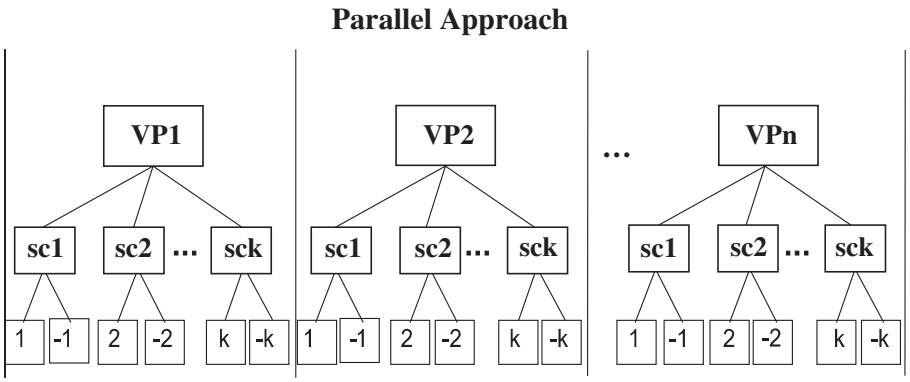


Figure 7. Data structures of the BTT simulator for parallel processing.

During the processing of the events, we must recognize the bogus termination-events that refer to calls marked as lost and the events that are beyond the interval's end. In the former case we simply skip the events. In the latter case the events are stored for processing during the next interval. A shift of all these events from the last positions of the time array to the beginning of the time array takes place, before filling it with the events of the next time interval. The corresponding auxiliary arrays are updated by a simple shift, too.

The structure of the proposed simulator is well fitted to parallel processing techniques for further savings of execution time. Figure 7 shows that the array-data (of Figure 5) can be generated in n parallel steps (as many as the number of VPs). Intuitively, this indicates that parallel processing can be applied on the data structures of the simulator.

5. PARALLEL PROCESSING

Based on the aforementioned data structures, we have implemented the simulator on a conventional, sequential computer [6]. It is worth mentioning that our implementation is based on the well-known IMSL FORTRAN library [20]. Then, on the sequential version of the simulator, we apply a parallelization process in order to generate its parallelized version, suitable for parallel computers, such as the SGI Origin with 4 processors, organized as 2 clusters of 2 processors each.

5.1. Dissociation between core simulation and built-in functions

First of all, we dissociate the core network simulation from other installed functions (as is VPB control) and we apply the parallelization process only on the core network simulation. As an example, Figure 8 shows the connections between the (core) simulator and the functions of VPB controller and BRT measurement.

The numbers in the core simulator correspond to the following part of the simulation:
(1) Initialization, stabilization, simulation up to the first control point.
(2) Waiting for new bandwidth distribution.
(3) Bandwidth rearrangement, BRT—new VPB allocation.
(4) Simulation until the next control point.

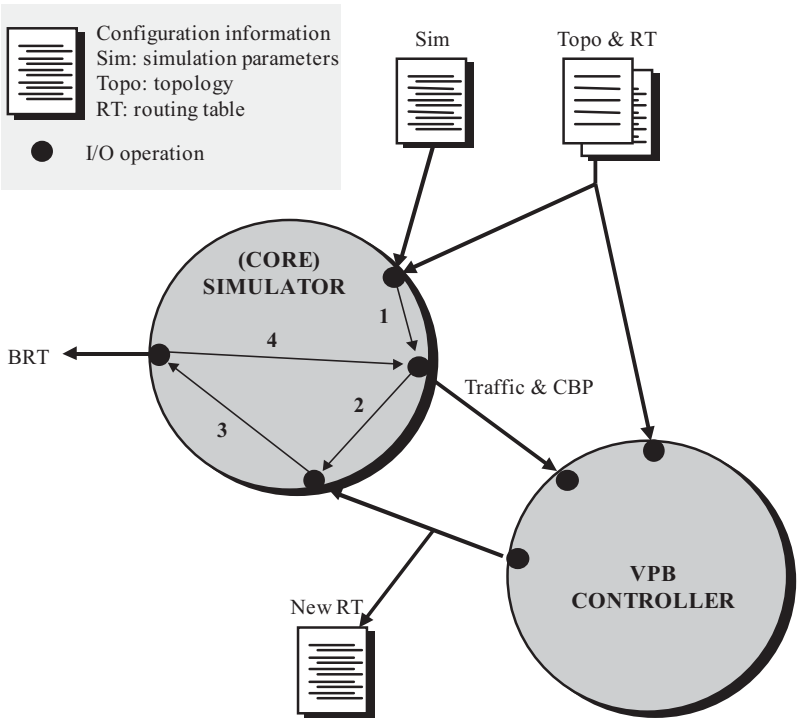


Figure 8. Core simulation and the functions of VPB control and BRT measurement.

The input data of the simulator and the controller is common. The simulator starts using the configuration information of the simulation (simulation parameters: total duration, small time-interval duration, time before stabilization, etc.), the network topology and the list of all VPs of the ATM network registered in the table, called *routing table* (RT). Then, the simulation runs (always in steps of time t) until the network stabilizes (that is, reaches the steady state of operation and, beyond this time point, statistical data of the simulation are valid and reliable) and the first control point is reached (1).

Beyond the stabilization time point, the simulation time is also divided in control intervals, which are larger than the small time-intervals t used for the batch processing (actually they are an integer multiple of t). The average carried traffic and CBP for each service-class and VP are measured during the control interval in the ATM switches. At the end of each control interval the simulator passes these measurements to the controller and waits for its response (2).

In return, the controller during its control interval calculates the offered traffic of each service-class and determines the new bandwidth allocation for all VPs registered in the routing table. The new routing table is passed to the simulator (i.e. to the ATM-switches) as soon as it is completed. The VPB rearrangement procedure starts at the beginning of the next control interval (3), but as previously described this will take some time, BRT, due to the already existing connections in the VPs whose bandwidth has to be reduced. When these connections are released, the new routing table is applied simultaneously to all VPs. In order to start the rearrangement procedure the controller sets a flag to pass this information to the switches. The flag is reset by that switching-pair (see Section 5.2.5) which has delayed most to get the assigned bandwidth, while waiting for a required number of connections to be released. In the meantime, all the switching-pairs, which have to reduce their bandwidth as the controller has ordered, set the free and released bandwidth in an idle state, unit by unit, until the bandwidth needed is achieved. Obviously, the time passed waiting for the flag to be reset is measured as the BRT. After this, the simulation may go on with the new bandwidth distribution until the next control point (4).

The simulation continues by cycling with the steps (2)–(4) until the simulation ends.

5.2. Possible methods for parallelization

The problem of software parallelization is basically an organization problem and the general strategy for its solution is either the control ruled execution or the data ruled execution [21]. In the following, we examine possible methods of exploiting parallelism, which could be used in the multiprocessor implementation of the proposed simulator and are usually taken into consideration in parallel designs [22–24]. The choice often depends on the structure of the manipulated data, the actions and their properties related to inter-dependencies, or whether they need to be in a specific sequence. On the other hand, the computer running the program may influence the strategy. The hardware characteristics may be important, especially regarding memory hierarchy structure, memory access patterns and inter-processor communication.

5.2.1. Pipeline. The first method is the *pipeline model* (Figure 9), which is sometimes well fitted for repetitive tasks on large amounts of data. The pipeline model aims at performing several actions at the same time. Therefore, each processor may have its own program code corresponding to a subset of the global actions to be performed on the data. Processors communicate with each other in order to synchronize or get results.

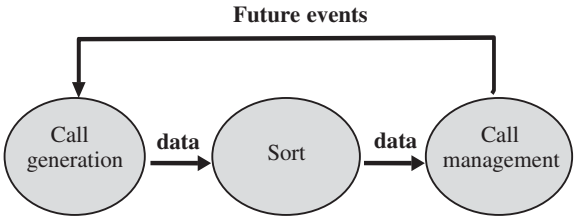


Figure 9. Pipeline model.

However, the pipeline model is not applicable here, because it would potentially raise two main problems:

(a) *Communications*. A large amount of data would have to be shared among the processors involved in the different stages of the pipeline. This includes at least the timetable array and the permutation array. There would also be the transfer stage when cycling at the end of the pipeline for handling a new interval. The events that are not processed because they occur too late, are included as part of the next processing stage. Actually, this point is not a crucial problem especially if the computer supports shared memory.

(b) *Load balancing*. A pipeline is by nature sequential; every stage expects data from the preceding one and sends its results to the next stage. Therefore, we must be careful about the amount of work that is to be executed at each stage. If some stages are over-utilized compared to others, they become bottlenecks. It results in other, less utilized stages, to be idle for long periods of time, due to lack of data. Of course, this degrades the overall global performance. The ideal state is to have a fully balanced pipeline, with no waiting time between stages and all processors to be fully utilized. This implies that we need to divide the work equally among the available processors. Unfortunately, this type of distribution is very difficult in this case, because the three steps are very different. Even if we could predict the load distribution at the design time, the compiler will apply optimization, which greatly influences the execution time of each step, thus invalidating the prediction. Moreover, applying the predicted load distribution on the available processors could be another difficulty.

5.2.2. Compiler optimizations. The second method for parallelization involves automated tools for code optimization. For example, Silicon Graphics provides tools for code parallelization on the Origin computer. By reading and compiling a sequential source code, the *MIPSpro Power Fortran 90* (or *Power Fortran*) compiler detects data dependencies, large loops and gives clues for the parallelization tasks [25]. It can also produce a modified source code ready for a parallel execution. The following remarks concern this solution:

- The automatic conversion keeps the original programming sequence and structure. If a program has not been designed for parallel execution, obviously, its performance would be far away from the optimal achieved by an appropriate design.
- The programmer needs to review the results and use them rather as guidelines than strict steps to follow, as the suggested changes may be negative in performance and readability, may raise errors or become useless.

- Learning how to use the automatic converter and how to review the results for optimality and correctness takes time, maybe more than to redesign the program from scratch.
- Finally, not all-parallel systems offer efficient, automatic parallelization tools today.

The strategy used by *Power Fortran* is to look for outermost loops and to evaluate their execution by means of several execution threads and what this implies for the related data. In this case, inline directives are added to the source to control processors and data distribution, as well as additional material, to justify the compiler's choices. Otherwise, the checking proceeds with the nested loops. As a result, the number of processors will change during execution, from one processor for sequential sections, up to several processors for parallelizable sections.

5.2.3. Global loops. Another interesting possibility is to find a solution by distributing data with a long running parallel execution. This will minimize the sequential ratio of the global execution time, providing thus better chances for optimality. This is also a solution, which is less machine-dependent.
Recall here, that calls are related to intervals, virtual paths (pairs of nodes) and service-classes. We can simplify the structure of the simulator as a 3-nested loop, one loop for each of these quantities. Since the number of service-classes is usually very small (possibly 2), service-classes are not suitable for parallelization. The entire design of the simulator excludes a partition based upon service-classes. As a matter of fact, we can note that the very different properties of the various service-classes would lead to dramatic contrast in the load of processors.

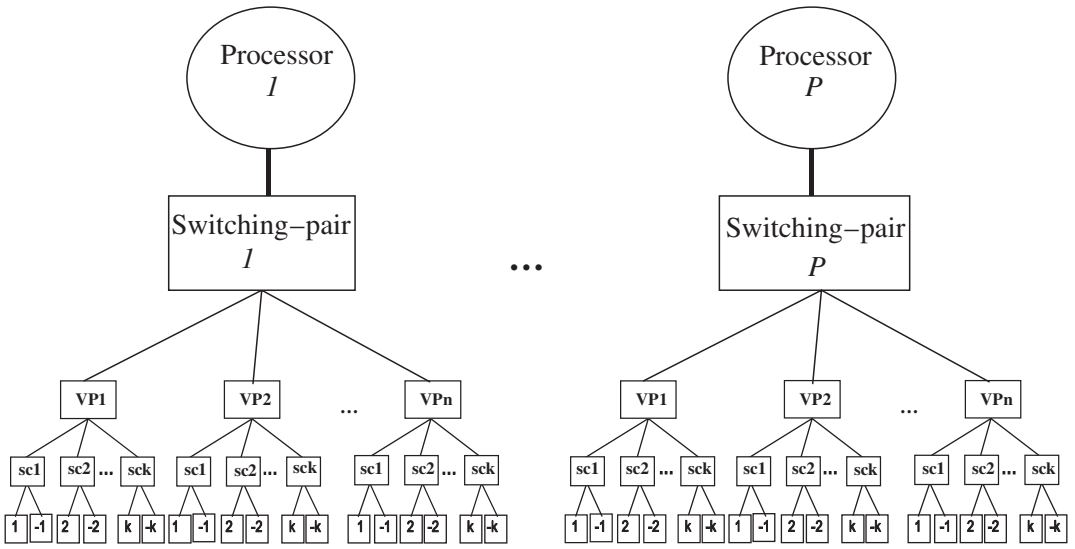
5.2.4. Interval loop. It is impossible to distribute the small intervals (t) on processors. Time is a capital point in the simulation while the time intervals are strongly tied together. There is no way to break their sequential order, because:

- Some events generated during one interval may actually occur in a following one.
- The occupied capacity of the network will depend on these events. This is valid for non-terminated calls.

Thus, only one processor will be able to work at a time and all the others will be waiting for another one to end the processing of an interval. This is no better than a sequential run.

5.2.5. Switching-pair or VP loop. A telecom network consists of nodes and links. Let us consider a network of D terminal nodes. Then, by distinguishing the traffic-flow directions we have $D^*(D-1)$ communicating terminal-node pairs, called *switching-pairs*. From an operational point of view, the switching-pairs are independent from each other and they can be assigned to processors. With an one-to-one assignment, $P = D^*(D-1)$ processors are needed (see Figure 10).
A switching-pair may utilize one or more virtual paths (VPs). Once the VPs of some switching-pair are defined, they are operationally independent. Therefore, instead of distributing the switching-pairs to processors we can distribute the VPs. However, by assigning one processor to one VP, we need a number of processors, $n > P$ (see Figure 11).

Contrary to the small intervals, switching-pairs are independent. Distributing the switching-pairs to processors is a good solution to tackle the parallelization problem. Once the network configuration is defined and is stable, linked pairs of nodes work alone without the need to care for the others; the only limitation is the capacity they can use in the links. Thus, the simulator is composed of a group of processes, with each process corresponding to one switching-pair and



*Where P=D * (D–1)*

Figure 10. Distribution of switching-pairs to processors.

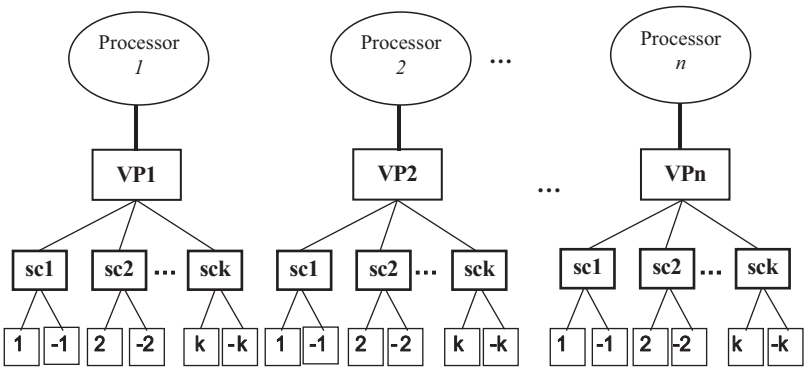


Figure 11. Distribution of VPs to processors.

simulating the network traffic for each small interval *t*. Actually, each processor handles a set of switching-pairs and runs a simulation on this subset of the network. Their synchronization however is especially important for calculation of simulation statistics and also for the built-in functions.

Instead of the switching-pair loops we may also consider *VP loops*. It is worth mentioning, however, that a switching-pair may not use only one VP per traffic flow direction but several. Still, in the latter case, each one of the VPs acts alone, once the routing pattern is applied. So we may consider the problem partitioning either with the VPs or with the switching-pairs, without important changes in the parallelization process.

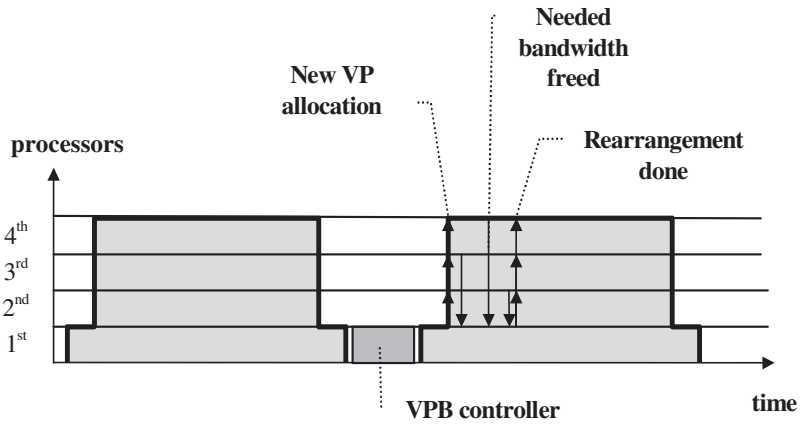


Figure 12. Parallel execution.

5.3. Choice far implementation. Our choice for this implementation uses the "*VP loop*" solution, where the VPs of the ATM network are distributed among the processors. This choice is more advantageous than the alternative solution of the 'switching-pair loop', since a fixed number of VPs exist in the network and each VP operates independently from the others. Thus, we have explained the validity of the data structures shown in Figure 7.

As an example, Figure 12 displays the execution frame of the simulator on 4 processors and the co-operation with a VPB controller. At the end of each control interval, one processor will gather data from all the others and will calculate input data for the controller. After waiting for it to terminate, it will then diffuse the new configuration data and the simulation will go on. The anticipated time-performance of this parallel processing scheme will be near to the one-fourth-time of that of a sequential processing scheme (by the same single-processor computer).

6. PERFORMANCE

The performance of the proposed batch-type, time-true ATM-network simulator is compared against that of a call-by-call time-true simulator in respect of required CPU-time and computer memory, when both simulators run on sequential computers. A first evaluation test with a 10-node network and realistic traffic load of the telephone service, showed that 10 h of simulation time are covered in 90 min CPU-time on a VAX-6330, while a call-by-call, pure time-true, network simulator spends at least five times as much time (depending on the network operations) [5].

Herein, we comparatively evaluate the performance of the BIT simulator against that of a call-by-call simulator, for a series of ATM networks of ring topology, with a size of 5–10 nodes. Two service-classes are accommodated in the networks: the telephone service with traffic 260 erl offered in each VP and a video service requiring 24 times more bandwidth per call than a telephone call (64 Kbps), with traffic 12 erl offered in each VP. The grade-of-service is 3% CBP for both service classes. Since we assume circuit emulation traffic, the bandwidth reservation

scheme, which equalizes the CBP between the service-classes, is considered. What is actually simulated is the traffic offered to the VPs, and what is measured is the CBP. The following figures reveal the performance of the simulators.

Figure 13 shows the execution time in seconds of the BTT simulator for 10 h of simulation time, versus the small time-interval t, which varies from 10 to 100 s.

Figure 14 comparatively shows the execution time of the call-by-call (CBC) simulator and the BTT simulator (shown in the horizontal axis, by its parameter t). The execution time of the BTT simulator is the same as that shown in Figure 13.

In Figure 15, we show how many times the BTT simulator is faster than the call-by-call simulator. This is denoted in the vertical axis of Figure 15 by the ratio of the execution times of the two simulators (execution time of call-by-call simulator divided by the execution time of BTT simulator).

Figure 16 shows the effect of the small time interval t in the performance (execution time) of the BTT simulator. As Figure 16 shows, the small time interval t varies from 1 to 13 min (in steps of 1 min). When t is 6 min the execution time is the shortest. If we consider some longer time for t than 13 min the execution time deteriorates.

The trade-off for the timesaving is the computer memory consumption. This is an obstacle that restricts the length of the interval t because, obviously, the required computer memory is increased as t increases. Figure 17 shows the length of the basic arrays of Figure 5 both for the call-by-call (CBC) and the BTT simulator. At least the double amount of memory (array length) is required for the BTT simulator in comparison with the call-by-call simulator when t is in the order of 100 s.

The following tests reveal the accuracy of the results obtained by the BTT simulator:

(a) We consider the aforementioned two service-classes in a single end-to-end link (VP connection—a 2-node network), which has a bandwidth capacity of 44.736 Mbps

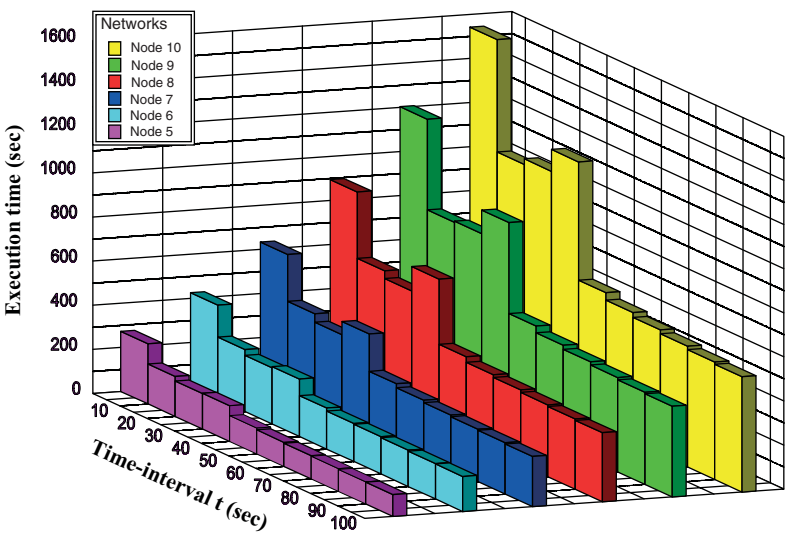


Figure 13. Execution time for the BTT simulator.

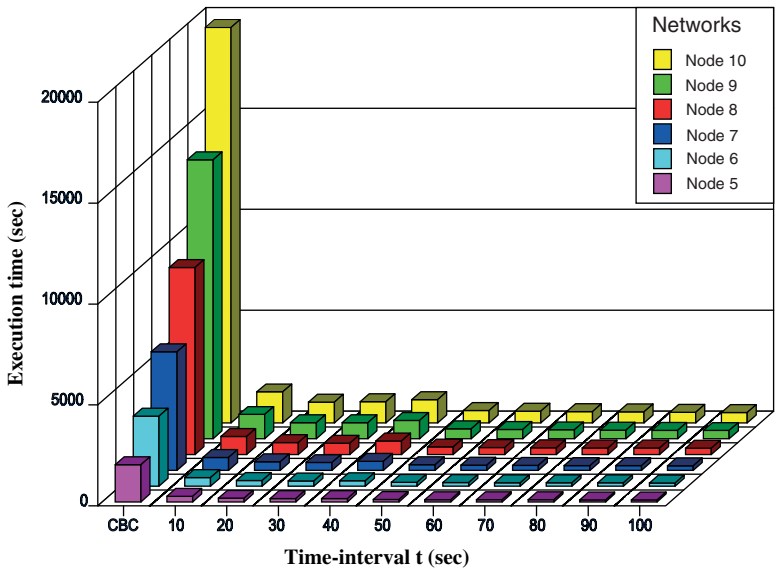


Figure 14. Execution time of BTT and call-by-call simulators.

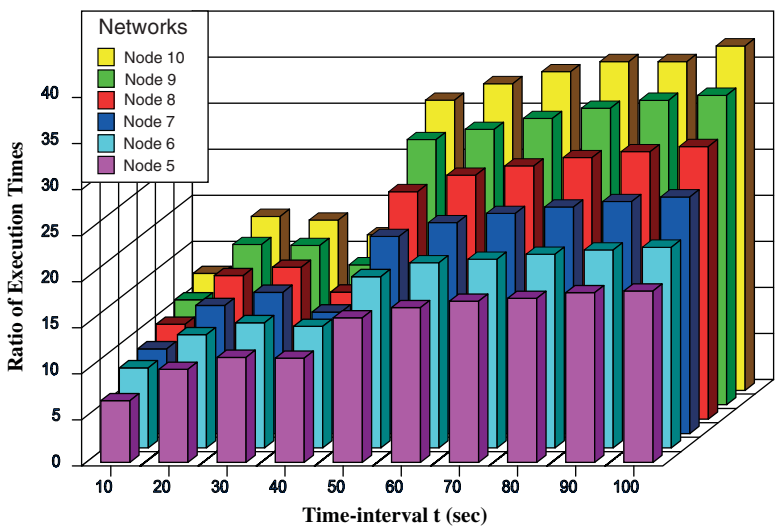


Figure 15. Comparison of execution times of the BTT and call-by-call simulators.

(= 699 * 64 Kbps). Calls of both service-classes arrive according to a Poisson process while their holding times follow the negative exponential distribution with the same average time of 100 s. By applying the well-known Erlang multi rate loss model (EMLM) we can estimate the CBP in this link for each service-class without considering any trunk

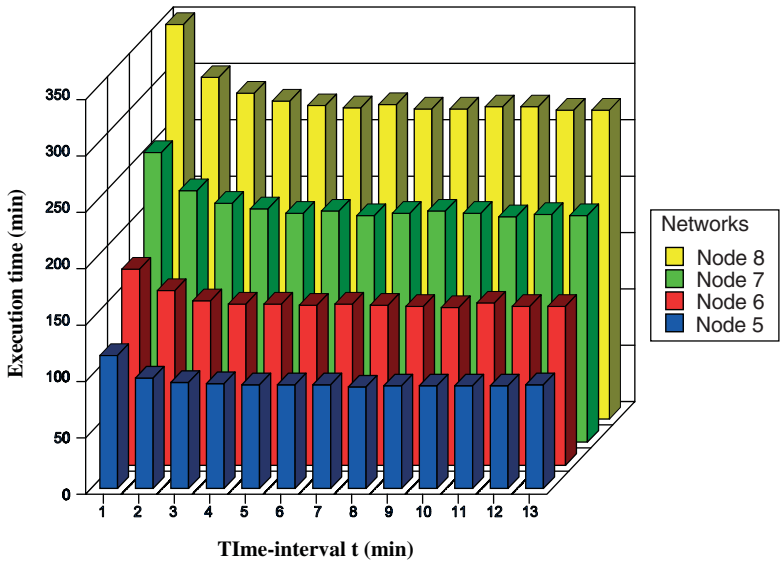


Figure 16. Effect of the time-interval t in the time-performance of BTT simulator.

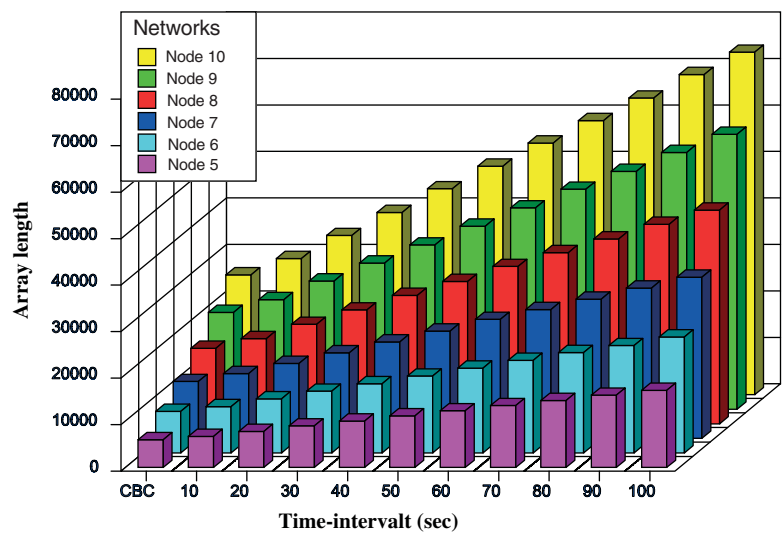


Figure 17. Computer memory requirements for the BTT and call-by-call simulators.

reservation scheme (in order for the EMLM to be accurate) [12,19]. We repeat the CBP calculations while we increase the offered traffic load of the telephone service by 100%, in steps of 20% (that is in steps of 52 erl, as $260 * 0.20 = 52$). Then, the CBP is measured by the BTT simulator with $t = 6$ min (small interval). Both the simulation and the EMLM results are presented in Table I, for comparison. The simulation results are mean

Table I. Accuracy of the BTT simulator—single link test.

Traffic load	1st service-class		2nd service-class	
increment (%)	EMLM CBP	BTT simulator CBP	EMLM CBP	BTT simulator CBP
0	0.00101	0.00108 ± 0.000320	0.02990	0.02983 ± 0.005314
20	0.00251	0.00283 ± 0.000466	0.06960	0.07183 ± 0.006986
40	0.00521	0.00497 ± 0.000367	0.13483	0.13480 ± 0.009757
60	0.00939	0.00952 ± 0.000841	0.22480	0.22856 ± 0.012478
80	0.01522	0.01524 ± 0.001227	0.33390	0.33429 ± 0.011953
100	0.02159	0.02300 ± 0.001424	0.43576	0.45035 ± 0.009486

values (of CBP) obtained according to the 'replication' method (8 runs of 8 h simulation time each) [1]. They are accompanied with confidence intervals of 95% [1,3]. Table I shows a high accuracy of the simulation results, since the EMLM is a well-established analytical model. It is worth-mentioning that the results of a reliable simulator are considered more accurate than that of analytical models.

(b) We consider a 10-node network consisting of end-to-end links of 44.736 Mbps each. The network accommodates the above-mentioned service-classes and traffic loads, without a trunk reservation scheme (in order to meet different CBP for each service-class and obtain more results for evaluation). Through simulation, we measure the maximum CBP among all switching node-pairs, for each service-class, and present average values (of maximum CBP) according to the replication method (again, 8 runs of 8 h each). Needless to say, the maximum CBP may occur in different switching pairs, therefore, higher values than that of the test in the single end-to-end link are anticipated. Our main concern, in this test, is to check the correctness of our hypothesis that the resultant CBP is independent of the small interval t. In other words, running the simulation with different small intervals (t), we must show that the CBP measurements were drawn from populations having the same mean (null hypothesis). To this end, we perform the classical analysis of variance, which requires the calculation of variance ''between'', V_B, and 'within', V_W, the set of measurements of maximum CBP [26]. If s_{ti} is the variance and x_{ti} is the mean of the measured maximum CBP (8 values for each offered traffic load) when the value of small interval at the ith set of measurements is t, the V_B and V_W are calculated as:

$$V_B = \sum_{i=1}^{k} n(x_{t_i} - x_{GM})^2/(k-1)$$

$$V_B = \sum_{i=1}^{k} s_{t_i}/k$$

where k is the number of sets of measurements, corresponding to the number of small intervals ($k = 3$), x_{GM} is the mean value of x_{ti}, $i = 1, 2, \ldots, k$, (grand mean) and n is the number of CBP measurements per small interval ($n = 8$ for each small interval).

Having calculated the V_B and V_W we calculate the ratio $F = V_B/V_W$, which is called F statistic and follows the F (Fisher) distribution with $\{(k-1), k(n-l)\}$ degrees of freedom. We reject our null hypothesis with 'level of significance' of 5% or 1% (that is, the probability of making

correct decision is 95% or 99%, respectively), if the F statistic is greater than the so-called critical value (threshold), $F_{95}^{\{(k-1),k(n-l)\}}$ or $F_{99}^{\{(k-1),k(n-l)\}}$, respectively. For $k = 3$ and $n = 8$, the critical values are obtained from the F distribution so that:

$$\text{Level of significance} = 5\%: \ \Pr\{F \leqslant F_{95}^{(2,21)}\} = 95\% \Rightarrow F_{95}^{(2,21)} = 3.47$$

$$\text{Level of significance} = 1\%: \ \Pr\{F \leqslant F_{99}^{(2,21)}\} = 99\% \Rightarrow F_{99}^{(2,21)} = 5.78$$

Figures 18 and 19 show the maximum CBP versus the offered traffic load increment of the 1st service-class, for the 1st and the 2nd service-class, respectively, when the considered three small intervals of the simulator are: $t_1 = 6$ min, $t_2 = 4$ min, $t_3 = 2$ min. The error bars represent confidence intervals of 95% for the calculation of the maximum CBP. For each traffic-load the F statistic is shown together with the probability of larger F (probability, P, in parenthesis), based on the three different small intervals. In the case of 1st service-class, only one null hypothesis is rejected either with level of significance 5% or 1% (when the traffic load increment is 20%), while another one is rejected only with level of significance 5% and is accepted with level of significance 1% (when the traffic load increment is 80%). For the 2nd service-class all null hypotheses are accepted with level of significance of 5% (and therefore with level of significance of 1%). These results are considered absolutely satisfactory.

Finally, we present an application example of network simulation including non-constant arrival rates and realistic traffic-load conditions. We consider again the 10-node network of the previous test. The 1st service-class remains the telephone service of 64 Kbps. The 2nd service-class is an elastic service-class, asking for 384 Kbps (bandwidth per call), but it can reduce the bandwidth requirements to 128 Kbps (in steps of 64 Kbps). One more elastic service-class is accommodated in the network that is capable of changing the bandwidth requirements per call

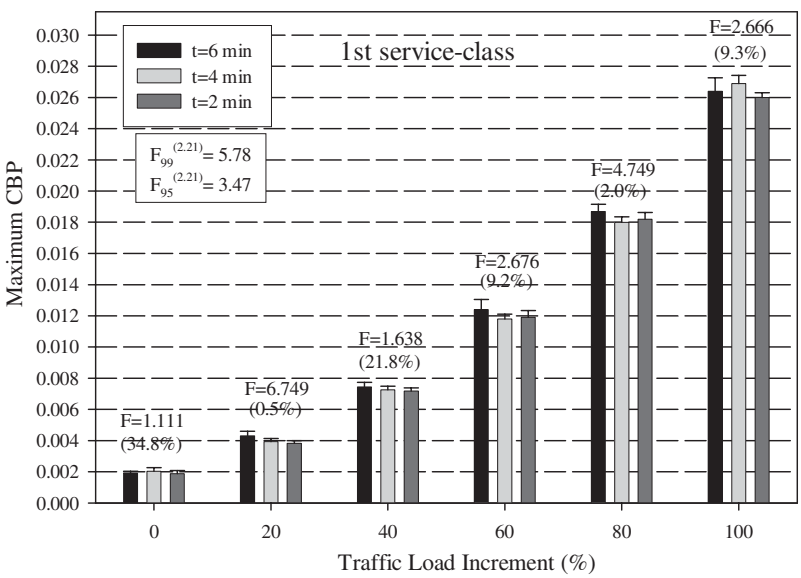


Figure 18. BTT simulator accuracy—maximum CBP and analysis of variance for the telephone service,in the 10-node network.

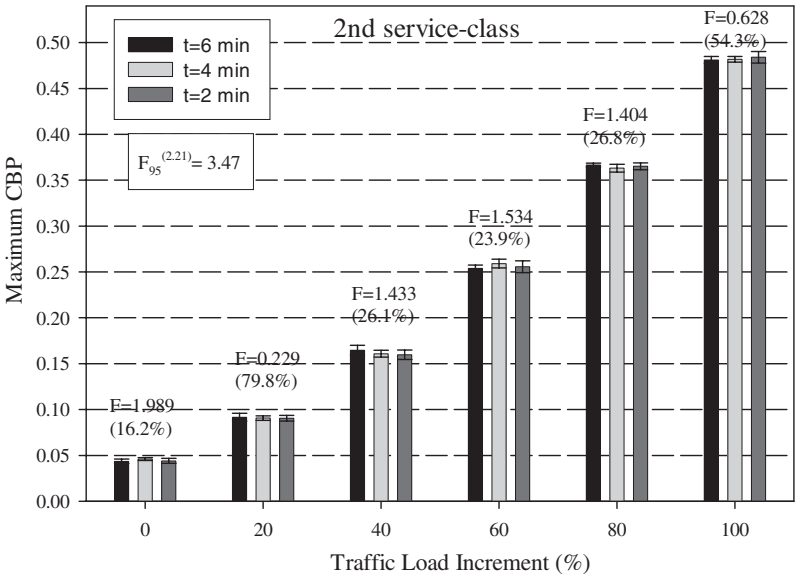


Figure 19. BTT simulator accuracy—maximum CBP and analysis of variance for the video service, in the 10-node network.

from 1536 to 768 Kbps. No trunk reservation scheme is applied among the service-classes. The initial offered traffic-load of the 1st service-class is 260 erl (in each VP). The 2nd and 3rd service-classes have the same offered traffic load of 12 erl each. We consider the following offered traffic-load conditions for a period of 6 h. The initial traffic-load remains constant for the first 2 h, then varies randomly (increases or decreases linearly) in all network's VP, by a maximum of $X\%$, reaching the maximum/minimum value after 2 h, and then remains constant again, for the next 2 h. By the BTT simulator, we measure the final maximum CBP among all switching node-pairs, for each service-class, and present average values (of maximum CBP) according to the replication method (8 runs), with confidence intervals of 95%. More precisely, if $A_0[i,j]$ $(i,j = 1,2,\ldots,10$ for the 10-node network) is the initial traffic-load, the final traffic-load $A_k[i,j] = A_0[i,j] * [1 - X(k) + 2 * X(k) * RND]$ where RND is a pseudorandom number from a uniform (0,1) distribution, and $X(k) = 20, 40, 60, 80$ and 100% for $k = 1, 2, \ldots, 5$, respectively. Figure 20 shows the results when the maximum traffic fluctuation X is $0, 20, 40, \ldots, 100\%$.

7. CONCLUSIONS

In this paper we have proposed a batch-type, time-true simulator, which results in considerable savings in execution time, compared to a call-by-call simulator. In our simulation method, the sorting operations are drastically reduced compared to the existing method of call-by-call simulation. The entire simulation duration is divided into N short time-intervals of equal duration t. During every small interval t, a batch processing of the call origination and termination events is executed and then the time points of these events are sorted. During the entire simulation time, N sorting executions are needed. The proposed simulator is a pure time-

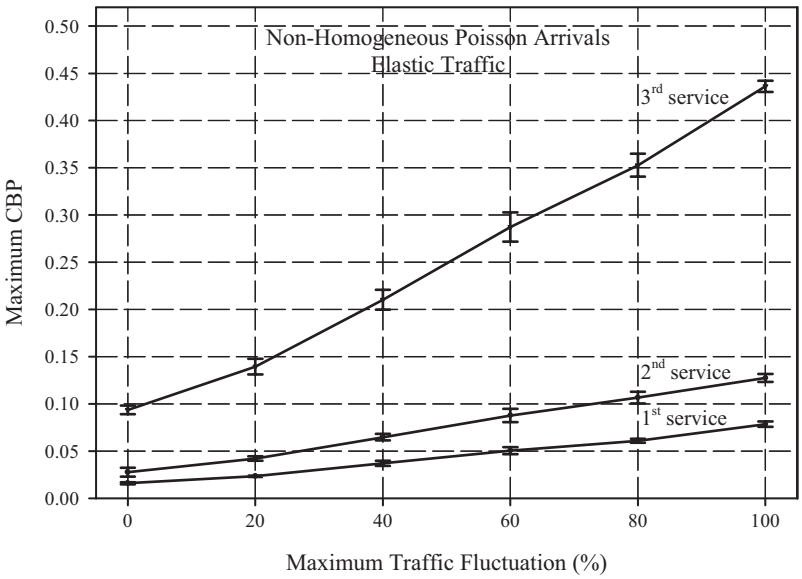


Figure 20. Maximum CBP in a 10-node network, under realistic traffic-load conditions.

tracing simulator and its main function is to apply network/traffic control schemes. Its data structures are well fitted to parallel processing techniques for further savings of execution time. We first implemented the simulator on a sequential computer and then we applied parallelization techniques to implement it on a parallel computer. In order to simplify the parallelization procedure, we dissociate the simulation from the actual network/traffic control problem. That is, the network/traffic control problem is not implemented in a parallel manner. The time-performance of the proposed simulator has been extensively compared against a call-by-call simulator, for a series of ATM networks. The accuracy of the obtained results from the BTT simulator has been tested.

ACKNOWLEDGEMENTS

Work supported partially by the Caratheodory programme of the University of Patras/Greece.

REFERENCES

1. Akimaru H, Kawashima K. *Teletraffic—Theory and Application*. Springer-Verlag: London, 1993.
2. Sykas E, Vlakos K, Venieris I, Protonotarios E. Simulative analysis of optimal resource allocation and routing in IBCN's. *IEEE Journal on Selected Areas in Communications* 1991; **9**(3):486–492.
3. Pitts JM, Schormanns JA. *Introduction to IP and ATM Design and Performance—With Applications Analysis Software* (2nd edn). Wiley: Chichester, U.K., 2000.
4. Kosten L. *Stochastic Theory of Service Systems*. Pergamon Press: Oxford, 1973.
5. Inoue A, Yamamoto H, Harada Y. An advanced large-scale simulation system for telecommunication networks with dynamic routing. *Proceedings of the NETWORKS* 89, September 1989; 77–82.

6. Logothetis M, Kokkinakis G. A batch-type time-true ATM-network simulator. *Proceedings of the COMCON5*, Rethemna/Crete, Greece, 1995; 231–241.
7. Riley GF, Fujimoto RM, Ammar MH. Ageneric framework for parallelization of network simulations. *Proceedings of the 7th International Symposium on Modeling. Analysis and Simulation of Computer and Telecommunication Systems*, *MASCOTS*'99, College Park, MD. October 1999.
8. ITU-T Recommendation 1.311, B-ISDN general network aspects. Geneva, 1991.
9. Sato K-I, Ohta S, Tokizawa I. Broad-band ATM network architecture based on virtual paths. *IEEE Transactions on Communications* 1990; **38**(8):1212–1222.
10. Moscholios I, Logothetis M. Call-level QoS assessment in ATM networks supporting elastic traffic. *Proceedings of the ICC 2001*, Helsinki, 11–15 June 2001.
11. Saito H, Kawashima K, Sato K-I. Traffic control technologies in ATM networks. *IEICE Transactions on Communications* 1991; **E74**(4):761–771.
12. Logothetis M, Shioda S. Centralized virtual path bandwidth allocation scheme for ATM networks. *IEICE Transactions on Communications* 1992; **E75-B**(10).
13. Logothetis M, Shioda S, Kokkinakis G. Optimal virtual path bandwidth management assuring network reliability. *Proceedings of the ICC*'93, Geneva, 1993.
14. Logothetis M, Kokkinakis G. Network planning based on virtual path bandwidth management. *International Journal of Communication Systems* 1995; **8**:143–153.
15. Logothetis M, Kokkinakis G. Influence of bandwidth rearrangement time on bandwidth control schemes. *Proceedings of the 4th International Conference on Advances in Communication & Control*, *COMCON*4, Rhodes, 1993.
16. Shioda S. Evaluating the performance of virtual path bandwidth control in ATM networks. *IEICE Transactions on Communications* 1994; **E77-B**(10):1175–1187.
17. Papanikos I, Logothetis M, Kokkinakis G. Virtual path bandwidth control versus dynamic routing control. *Performance Modelling and Evaluation of ATM Networks*. Chapman & Hall: London, U.K., 1996.
18. Logothetis M, Shioda S. Medium-term centralised virtual-path bandwidth control based on traffic measurements. *IEEE Transactions on Communications* 1995; **43**(10):2630–2640.
19. Moscholios I, Logothetis, M. Applicability of the extensions of the Erlang multirate loss model on ABR services in ATM networks. *Proceedings of the London Communication Symposium*, *LCS2000*, London, September 14–15, 2000.
20. IMSL, Visual Numerics Inc., Houston, http://www.vni.com/products/imsl.
21. Hambrusch SE. Models for parallel computation. *Proceedings of the International Conference on Parallel Processing Workshop*, *ICCP—Workshop* '96, 1996.
22. Koufopavlou OG, Tantawy AN, Zitterbart M. Analysis of TCP/IP high performance parallel implementations. *Proceedings of the IEEE Conference on Local Computer Networks*, Minneapolis, MN, September 1992.
23. Koufopavlou OG, Zitterbart M. Parallel TCP for high performance communication subsystems. *Proceedings of the GLOBECOM '92* 1992; 1395–1399.
24. Zitterbart, M. Parallel protocols implementations on transputers. *Proceedings of the IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Tucson, AZ, February 1992.
25. Loveman DB. Fortran: a modern standard programming language for parallel scalable high performance technical computing. *Proceedings of the International Conference on Parallel Processing Workshop*, *ICCP—Workshop* '96, 1996.
26. Jandel SigmatStat—Statistical software, http://www.jandel.com/SigmaStat/index.cfm.

AUTHORS' BIOGRAPHIES

Michael D. Logothetis was born in Stenies/Andros, Greece, in 1959. He received his Dipl-Eng. degree and PhD in Electrical Engineering, both from the University of Patras, Patras, Greece, in 1981 and 1990, respectively. From 1982 to 1990, he was a Teaching and Research Assistant at the Laboratory of Wire Communications, University of Patras, and participated in many national and EC research programs, dealing with telecommunication networks, as well as with natural language processing. From 1991 to 1992 he was Research Associate in NTT's Telecommunication Networks Laboratories (Tokyo). Afterwards, he was a Lecturer in the Department of Electrical & Computer Engineering of the University of Patras, and since 1996 he is an Assistant Professor in the same department. His research interests include traffic control, network management, simulation and performance analysis and optimization of telecommunications networks. He is a member of the IEEE (Commun. Society—CNOM), IEICE and the Technical Chamber of Greece.

Dr Liotopoulos has graduated from the University of Patras/Computer Engineering Department, in 1988 and received his MSc and PhD degrees from the Computer Sciences and Electrical & Computer Engineering Department of the University of Wisconsin-Madison, USA, in 1991 and 1996. Currently, he works as a Researcher and R&D Unit Manager at the Research Academic Computer Technology Institute (C.T.I.), in Athens, Greece. He also teaches at the Greek Open University. His research interests include digital communications, broadband switching, electro-optical switching, ATM networks, parallel & distributed architectures, multi-processor/multi-computer systems, performance analysis and design optimization. Dr Liotopoulos is a member of IEEE, ACM, AMS, Sigma–Xi, Eta–Kappa–Nu, the Technical Chamber of Greece, the IEEE Technical Committee of Communications Switching & Routing and the Editorial board of Wireless Communications Mobile Computing Journal.