A Study of the Performance Simulation of a File-Server-Configured Micro-Lan

Paul Roth
Ilan Katz

TR 87-9

February 25, 1987

TABLE OF CONTENTS

INTRODUCTION AND OBJECTIVES

This document is a final technical report on a research study that was conducted by Virginia Tech under joint grants from the Virginia Center for Innovative Technology and the National Bureau of Standards in cooperation with the Applied Research Section, Internal Reserve Service.

The research program incorporated a pilot program to develop simulation-based performance evaluation techniques for Micro-LAN systems, with the objective of predicting network performance characteristics such as response time and CPU utilization, under varying traffic conditions and different LAN configurations.

The main quantitative objectives of this research were:

1. To investigate performance characteristics of a micro-LAN system, utilizing discrete-event simulation techniques.

2. To develop a prototype modeling tool for use by system managers and/or designers that would deliver reliable predictions of system performance.

3. To increase the awareness of the micro-LAN user community to the possibilities of using discrete-event simulation techniques as a configuration design tool.

SUMMARY OF CONCLUSIONS

1. Modeling and Simulation is Feasible
   A simulation model of a star-configured PC-LAN employing the Novell NetWare file-server operating system was synthesized and verified through a program of benchmarking runs using the actual system and software in configurations of up to 11 workstations. In other words, the model reliably predicted file-server network performance.

2. File Server Model Has General Applicability
   The simulation model predicts the response of the file server to a stream of read and write requests produced by a workstation's local program which is transparent to the file server. The model appears to be able to process any stream or burst of requests which is generated by an appropriate model of the workstation local processing program, e.g., DBMS, Lotus, etc. Further work to refine this "black-box" use of the model is suggested.

3. Model Has Led to Insights into System Behavior

Observed Behavior: Response-Time Linearity

We found that, in every instance observed, the average read or write response time increases linearly with respect to increasing numbers of workstations, when the request rate of each workstation is the same. We have termed this phenomenon "line-blocking"; the observed behavior can be explained by NetWare communication-handling features.

Hypothesized Behavior: Hierarchical Bottlenecking

From the behavior of the system, both modeled and actual, certain observations have led us to hypothesize the existence of a hierarchy of system bottlenecks, in which the system appears to enter "regions" where the response time rate-of-increase incurs an abrupt change of slope, and which are assumed to be caused by the response time being affected by a component which is bottlenecked by the workload rate of requests. It is suggested that, in hierarchical order, these components are (1) server disk, (2) server cache/CPU, and (3) network communications. Other observations suggest that the communications bandwidth characteristics may preclude it from constituting a major bottlenecking factor. Further research should be conducted to verify this far-reaching hypothesis.

4. Model Predictions are Generalizable

The simulation model was designed to predict the average response time of the star-configured NetWare server to requests generated by remote workstations. While the initial experiments dealt with a saturation-level workload (i.e., continuous running, with no human "think-time" pauses), analysis has lead to the postulation that pauses or interruptions in the work stream may have the effect of shifting the response curve to reflect improved response resulting from the decongestion of the system due to the pauses acting to lower the net workload rate. This suggests the hypothesis that a single model might serve to represent a variety of configurations, where the workload stream represents the variable quantity. Experiments with the model confirmed this behavior; experiments with a benchmark system need to be performed.

Another observation concerned a benchmark experiment with a large number of workstations connected to the server by a bus network, rather than a star. Measurements taken indicate the same linear response pattern as that of the star. Also, comparison of the measured response with that of the simulated bus system indicated a close correspondence of attained values in the linear response region of each system. What is suggested from these data is the hypothesis

that the network effects on the system response is small in the regions explored; this corresponds to the "hierarchy" hypothesis above. This also suggests the possibility that one simulation model may be usable over a range of network topologies. Further investigation may verify these hypotheses.

5. The "External" Model Domain is Sufficient for Performance Predictions

The performance results were attained with a so-called "external" system model: this incorporates a high-level description of the PC-LAN (NetWare) operating system as a component in the request execution stream. The high-level characteristics were abstracted from a detailed simulation of the operating system, which represented a detailed level of phenomena: this was termed the "internal model". The abstractions were employed in the "external" model under the hypothesis that low level features would have little effect on overall response time. So far this has held true, but should be explored more thoroughly.

RESEARCH PLAN

The initial phase of the research was a literature search conducted to assess the state-of-the-art of LAN modeling and simulation. A bibliography was generated.

The first modeling phase had, as its object, development of a detailed discrete-event simulation model of the Novell NetWare operating system. This model could be used to generate system performance data at the operating system level; its synthesis would provide enlightenment on the internal operation of the system. The model, termed the "internal model", was created with the help of the NetWare designers, who provided information at two sessions at the Novell, Inc. plant. As it turned out, the utility of the model was limited, but the insight gained into system operation as a result of the modeling process was invaluable in the development of a second model. The utility was limited because timing data on the operating system was unobtainable either from Novell or through any laboratory measurements feasible under our research plan. Therefore, though the operating system detailed model was not used per se, it did influence the results of the research.

The second modeling phase had, as its object, development of a discrete-event simulation model of a NetWare file server system, operating in a network environment. The functions of the detailed operating system were to be abstracted as a "black box" representation of the detailed model; this model was to process work units of realistic bursts of requests and to show the

sensitivity to such variables as number of terminals, "think" time, etc. This model, termed the "external model", was implemented in SIMSCRIPT II.5. Again, the Novell designers were helpful in providing design information. This model was calibrated through a program of combined benchmark experiments coupled with model tuning.

The benchmark phase of the program resulted in the accrual of experimental data which was used to tune the simulation model. It comprised of a series of planned, intensive network-running sessions at appropriately-configured network installations. Both grantor and grantee personnel contributed heavily to these sessions, some conducted in the Washington, D.C., area, some in Ogden, Utah. The experiment plans called for obtaining performance timings from system configurations which varied as to number of nodes, size and type of system workload, memory configuration, etc.

The documentation of the project was accomplished in two phases. An interim report was produced to cover the design of the "internal" model and report the literature survey. This final report documents the major effort, that of the "external" model and its results, but also includes elements of the interim report. A separate study, also included under this project, but conducted independently, involves the application of the SIMSCRIPT II.5 "external" model, reported herein, as a processor for request streams generated by particular terminal software. It is reported in the Interim Report.

MODEL DEVELOPMENT

System Description

The research encompassed a specific micro-LAN system which was arranged in a file-server configuration, where independent microcomputers store and retrieve data from a central node called a file server. The research was dedicated to a specific file server module: the Novell "NetWare" operating system. The conduct of this research included meetings with Novell, Inc., technical staff, to provide an in-depth understanding of the operating system, far beyond what might be obtained from the literature (commercial or scientific). The knowledge and understanding gathered in this process were then transformed into a flow-chart, functional (block diagram) model which describes network request-processing within the file-server. This model was later implemented in GPSS as an internal model of NetWare as a network operating system.

A more general model that encompassed the file server, workstations, and communication channels was then constructed and implemented in SIMSCRIPT II.5. It was later refined and tuned to reflect performance parameters based on benchmarking results derived from experimental runs with a real system. Because of the large variety of configurations, topologies, and versions in which NetWare appears, the model was tuned to a specific version of the operating system which is implemented as a "star" configuration using Novell's S/68000 hardware-software system as a file server.

This study was limited to those functions of NetWare that had a fundamental effect on system performance: Read and Write requests. Other functions such as data-integrity, security and file opening-and-closing were not included in the scope of the model.

The hardware configuration of the system is shown in Fig. 1 and consists of a central file server which regulates storage and retrieval of information from the disk system and also manages network services such as file spooling for printout, and gateways to remote mainframes or networks. The file server is connected through dedicated communication cards and channels to microcomputers which function as independent workstations.

The central file server is a Novell 68B-type server, designed to support the Novell S-Net topology. Up to 24 microcomputers can be attached to the server module with individual cables. The server's main CPU is a Motorola MC68000 processor running at 8 MHz clock-rate, able to access up to 8 Mbytes of main memory. Communication with workstations is managed by dedicated Network Interface Cards (NICs) built around Motorola M68B03 microprocessors. Similar NIC boards are installed at each workstation and communication is maintained at a rate of 500 K bits-per-second using a dual twisted-pair line.

The NetWare software configuration is shown in Fig. 2. It consists of four major software components: workstation and file-server operating system, workstation network interface (referred to as "NetWare Shell"), file-server operating system; and network communication utilities.

The workstation's operating system can be any version of MS-DOS. The NetWare Shell intercepts all calls directed from the application to the local DOS, transfers the local calls to DOS, and processes the others into file-server request packets which are then sent to the file server. Because of its DOS compatibility, the NetWare shell makes all network operations transparent to the executing application.

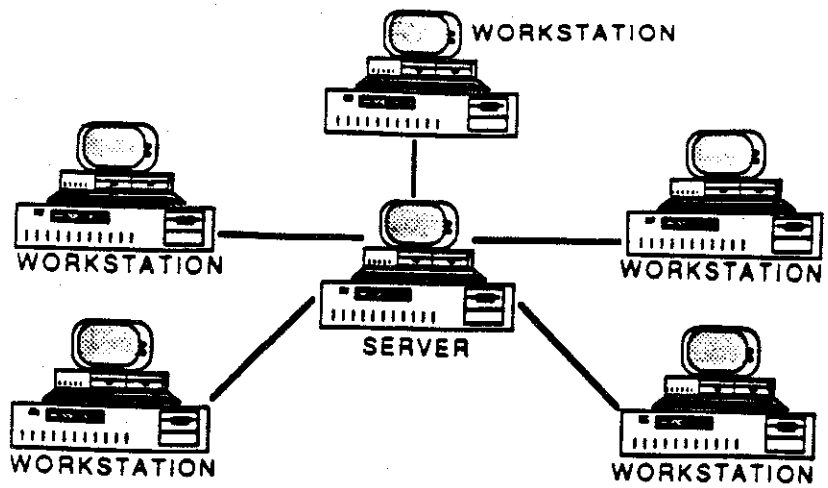

Figure 1 File-Server LAN: Star Configuration

Basic communication between the file server and the workstations is managed by utilities that implement the NetWare File Service Protocol (NFSP) provided by NetWare, which allows the shell to form, send, and receive communication packets. In normal operation the file-server only sees streams or bursts of "read" or "write" request packets which have been generated by the workstation shell.

The main component of the network software is the file-server operating system (referred to in Fig. 2 as "File Server Software"). The file-server O.S. is multi-tasking and multi-threaded, which means that one task does not have to run to completion before another may begin, thus preventing idle intervals during slow operations such as disk access, while other tasks are waiting to be serviced. Tasks are given different priority levels, allowing more important tasks to be serviced quicker.

External Process

The system can be described as having two processing cycles: an "external" cycle which includes the workstation, communication channel, file-server and disk system; and an "internal" cycle which encompasses the processing of requests once inside the file server.

Fig. 3 depicts the "external" cycle request path into the server. It starts with the workstation application submitting a DOS-like network request, which after some local processing, is turned into a request packet. It is transferred to the local NIC card, and then transmitted through a dedicated channel (on the "star" configured system) to the NIC card on the file-server. The next step involves communication between the NIC card and NetWare, after which, actual processing of the request (as described by the "internal" model) begins. Once a reply packet is ready it travels back the same way: from main memory to the NIC board, then transmitted to the requesting station's NIC board from where it is transferred to the workstation's main memory with appropriate notice to the shell which can now resume operation of the calling application process.

Internal Process

The "internal", or operating system, process execution cycle encompasses the possible processing paths a request may follow before being sent back as a reply. A flow-chart and further details are contained in the Interim Report.

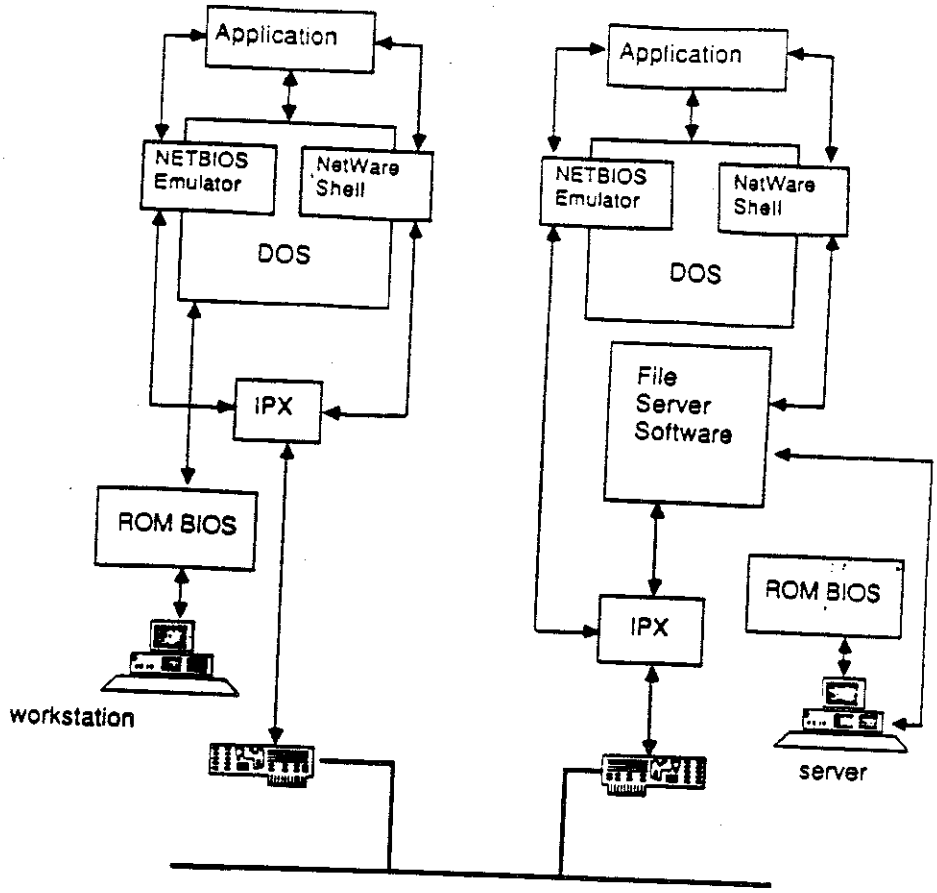

Figure 2: NetWare Software Architecture

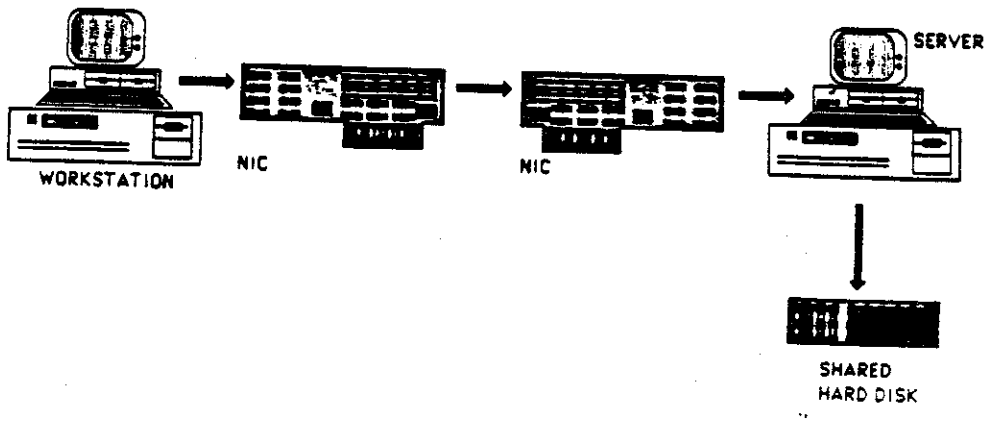

Figure 3: External Cycle - Request Data Path

The initial position of the request is as a packet stored in the NIC's memory. An operating system routine called the "polling process" scans these buffers in a round-robin fashion and if a request is found waiting, it is read into main memory. Once in main memory, the request is assigned to a "server process" which actually keeps track of the status and location of the request and runs in parallel, i.e., competes, with other server processes, for the resources required to answer this request. The number of server processes is limited to 8 and the polling process stops its scanning cycle if all 8 of them are busy, and resumes when at least one of them becomes free again. The polling process, being of lower priority than the server processes, must wait for the CPU until server processes no longer request the CPU.

As a first step of its processing, the request is decoded and interpreted to be either a "read" or a "write" request. Using the information provided by the requesting station, a unique file-record address is formed. The next step of processing depends on the type of request and the location of the specified record. In the case of a read request where the record resides in disk, the server process submits a request to another process, called the "disk process", to fetch the required record from the disk, by writing the request parameters into an area in memory which serves as the disk queue. The requesting server process then suspends itself until disk operation is complete and the desired record is brought to a special area of main memory, termed "cache" (not a hierarchical, high-speed "cache"), where it remains until eventually overwritten by another record.

If the desired record is already in cache memory, the activation of the disk process and the whole disk cycle are eliminated and the server process may proceed with the processing, uninterrupted to completion. The policy of keeping the last-referenced records in main memory to prevent additional disk accesses is termed as "caching". The last step in processing the read request is to form the reply packet and send it back to the appropriate NIC for transmission to the calling workstation.

For a write request, location of the desired record is identified first. If it is found in cache, the server process writes the data included in the request packet into this cache block and marks it as a "dirty" block. A reply packet is formed and sent immediately to the calling workstation. The "disk write" process will take care of copying the modified record from cache to disk. This process doesn't take place immediately. In cases of only slight modification, say 10 percent of a cache block, the operating systems "believes" that more modifications are soon expected and will refrain from writing that record to disk. In any case, if 3 seconds elapse without further changes made, this record is put into the disk write queue.

In case of a write request to a record which is not in cache at the moment, a cache block is selected according to a "Least Recently Used" algorithm to accommodate the new data. The selected cache block may, however, be dirty, itself. This means it was recently modified but not yet copied to the disk. In such a case the selected block must be cleaned first by immediately writing it to the disk, invoking for this purpose another process called "write-now". Meanwhile the server process that handles the original write request is suspended. Only after the block is clean can the server-process overlay it. A reply packet is then formed acknowledging the receipt of the packet, and the write process will then take care of writing the newly modified block to disk, according to the same rules described above.

An important system parameter is the amount of available cache memory: the more available, the greater are the chances that a record will be retrieved without accessing the disk, thus achieving shorter response times. Another significant parameter is the size of cache blocks in main memory. The relation to response time is generally more complicated in this case and depends on the type of application running on the network (large, contiguous files vs. small, random queries).

Disk access, as the slowest step in request processing, was given special effort to make it as efficient as possible. The disk queue is sorted according to the relative distance of the required record from the disk-head position and an "elevator seek" mechanism is applied to control the head's movement.

MEASUREMENTS AND BENCHMARKING METHODS

As with any modeling effort, a key element for describing a complete picture of the system is getting reliable, accurate timing information that, together with the knowledge of the system's sequence of execution, will enable the scaling and validation of the execution model to yield meaningful and reliable predictions of its behavior under different loads, configurations, etc.

NetWare's construction and installation precluded the direct measurement of execution times. No provisions were made by Novell to make "patches" to the operating system in order to "time-stamp" the execution time of a request or segments of it. The options to perform measurement were either: get a rough estimation by scanning the code and "hand-time" the instructions required to perform a certain task; or make external measurements using a benchmarking procedure that would run on the workstations. The second option, corresponding in level-of-detail to the "external" operation, was chosen, and a special purpose benchmarking program was designed, written and executed in order

to obtain end-to-end response times of requests. Appendix A contains the listing of the BASIC benchmarking program.

Since the study concentrated on the performance of read and write requests, the main idea of the benchmark program was to write and read to/from files in the file-server's memory in a repetitive fashion. By using the IBM PC DOS timing utility to measure the response time of a burst of requests (all of the same type), and then dividing by the number of requests, the average response time-per-request was obtained. By repeating the experiments for local and network operations and for different types of requests (read from a private file in cache, or from a shared file stored in disk, randomly or sequentially, etc.), some inferences could be made to quantify the time durations spent in: local processing of the request; communications; file-server CPU utilization and disk access.

One of the objectives of the benchmarking process was also to define the performance envelope, or capacity, of the network for different types of requests as outlined above. For this reason the measurements were conducted in "saturation" mode: the maximum request generation rate that individual workstations could produce. Capacity limit could be reached by adding more stations to the experiment until some bottleneck was reached. The definition of capacity limits would enable system designers to extrapolate these numbers, according to the intended application workload profile, and thereby obtain estimation of the saturation response time of a certain configuration.

Since "saturation" represents a worst-case workload condition, any realistic definition of workload would certainly cause an expected improvement in response time performance, due to lessening of system congestion, resulting from inter-burst and user time delays.

Two other system parameters besides response time were also recorded during the course of the experiments. CPU utilization and disk-queue length are two parameters sensed by NetWare and displayed on the monitor screen: they are updated on intervals of 1-second and represent the average value during the last interval. These parameters were found very useful in the validation process and in understanding the operation of the operating system during execution. A sample of measurement output is showed in Appendix A.

Experiments were conducted at the following facilities:

1. IRS Applied Research Branch, Alexandria, VA: 6-node LAN of mixed PC-compatible workstations (TI's, IBM-PC's, and IBM-AT), using Novell S-Net(star) for communication;

2. IRS Service Center, Ogden, Utah: 10-node LAN comprised of IBM PC's, XT's and a Compaq, using S-Net;

3. Novell, Inc., Service Center at Tyson's Corner, VA: 11-node LAN, all (except one AT) IBM PC's, using S-Net;

4. Virginia Tech Computer Science Dept., Falls Church, VA: 3-node PC network, bus topology, using an IBM AT as a server host, and Orchid PC/Net for communication.

5. HCI, Inc., Reston, VA: 30-node system of mixed workstations (mostly PC-compatible clones), using an Arcnet communication bus with a Novell 286B (bus) file-server.

The main problems encountered in this undertaking was the inability to secure the use of a large, homogeneous network, which would have enhanced the controllability of the experiment. In this context, some general observations can be drawn before describing the actual benchmarking results:

1. Though they showed similar behavior, Networks 1-3 exhibited slightly (about 15 percent) different results which may be attributed to the differences in hardware modules for the file-server and the disk system.

2. Different workstations yielded different results: an AT, for example, always gave the most variable results. Some possible reasons may be different BIOS versions or CPU-to-NIC board mechanisms.

3. The experimental network acquired under the grant, at the CS department, though very useful as a development tool, was not adequate for full-scale benchmarking experiments because of the small number of nodes and the wide range of results attributed to communication inefficiency (lost packets) of the Orchid interface boards.

The benchmarking experiments were designed so that operation of different subsystems could be isolated as much as possible. These experiments included:

1. Reading sequentially from a private file totally in cache.

2. Writing randomly to a private file totally in cache.

3. Repeating the above with different sizes of records (128 and 512 bytes).

MODELING APPROACH AND METHODOLOGY

The design and development of the "external" model took place in parallel with the benchmarking program. Both modeling and benchmarking evolved in parallel as more facts were gathered and as the model was refined by more understanding of the system's operation in a feedback process.

The initial approach was to build a detailed "internal" model of the file-server operating system detached from the external environment (communications and workstations). Since the file server is the central component in this system, it was anticipated that a detailed model would provide response time distributions which could later be implemented as a "black box" component in a broader system model to include its external environment. The internal model was implemented in GPSS/H and GPSS-PC: a listing of its current version is shown in Appendix B.

In the internal model, requests from workstations are viewed as a single incoming stream where only an assigned parameter signifies the "source" of the request. This approach was taken under the assumption that the file server and its request stream can be modeled as a Poisson-arrival queuing system. The model includes all major operating system processes described previously. Cache is implemented as an array where each cell represents a cache block and its associated dirty bit. Requests specify a record number which serves as the request's address, and the processing follows the description outlined in the System Description Section, except that the LRU page-replacement algorithm was not modeled. A random cache-block replacement algorithm was used instead, since no data on locality of requests was available. Record numbers specified in the requests where assumed to be uniformly distributed over the range of possible records. The internal model is driven by parameters such as request arrival rate, the percentage of read/write requests, cache size, and the size of requested records. A sample output is also provided in Appendix B.

The main problem faced during the course of developing the internal model was validation. No data or measurement techniques were available to determine internal execution times. The model was, therefore, parametrically driven. That means that, in its current state, this model can be used only to look at the file server's behavior sensitivity rather than to provide calibrated predictions of its performance. However, the sensitivity observed from performance experiments have given much insight into the behavior of the internal module when being tested as a modular component of a higher level model.

Another problem concerned the implementation of complex process interaction operations, which occur in NetWare. These problems, among others, dictated an approach change towards the concept of a general model at a higher level of description, to include, as modules, not only the file-server, but also communications and local processing. Internal processes would be modeled to a level of detail to the point where further detail would not affect the system's performance sensitivity, as seen from the point-of-view of the user or an application running on a workstation. For example, a detailed representation of the polling process is eliminated, and a representative polling time is lumped into the total CPU time required to process a request. Similarly, cache is not explicitly implemented: the disk-access rate is governed by a hit-ratio which is defined as an input parameter to the model. This model, implemented in SIMSCRIPT II.5, was validated against the benchmarking results. An additional version of the model was constructed for bus topologies which implement the CSMA/CD MAC schema (EtherNet). This model however was not validated because of a lack of a LAN system on which to perform benchmark experiments.

The "external" model was designed so that it could be validated and refined by the experimental results derived by the benchmarking program. Thus, it reflects a similar sequence of operations to the benchmarking program which involves the entire system (workstations and the file-server). Figures 4 and 5 illustrate the external process from two aspects. In Figure 4, the process 'burst' sequence is depicted by a burst of N requests sent from the workstation to the file-server (either a "read" or a "write" request). Since workstations are blocked between requests, these requests are sent sequentially, after an inter-request local processing time. After the burst is finished, the workstations are silent for some time, which can be considered as a longer local processing time or as the user's "think time", before a new cycle of request-bursts begins.

The process burst sequence depicted in Figure 4, controlled the frequency of requests presented to the system. This frequency was usually specified at a value sufficient to cause the system to operate at or near "saturation" during most experimental situations.

Figure 5, describes in greater detail the processing procedure of a single read request, and may be considered as an enlargement of the "request" block in Figure 4. A request is originated by a workstation after local processing time which produces a request packet indicating the type of request and the originating station. The request packet is then sent to the file server over the communication line, an operation which also consumes a certain amount of time. Upon receipt at the file-server, the packet is stored in a queue until its turn comes. At this point, it is

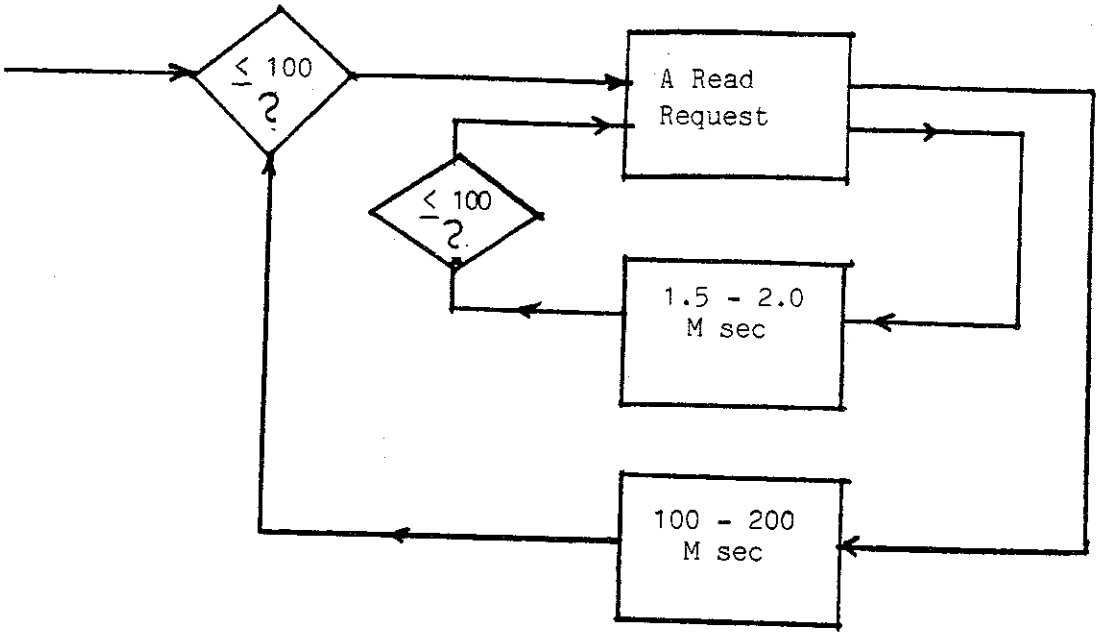

Figure 4: Process Burst Sequence

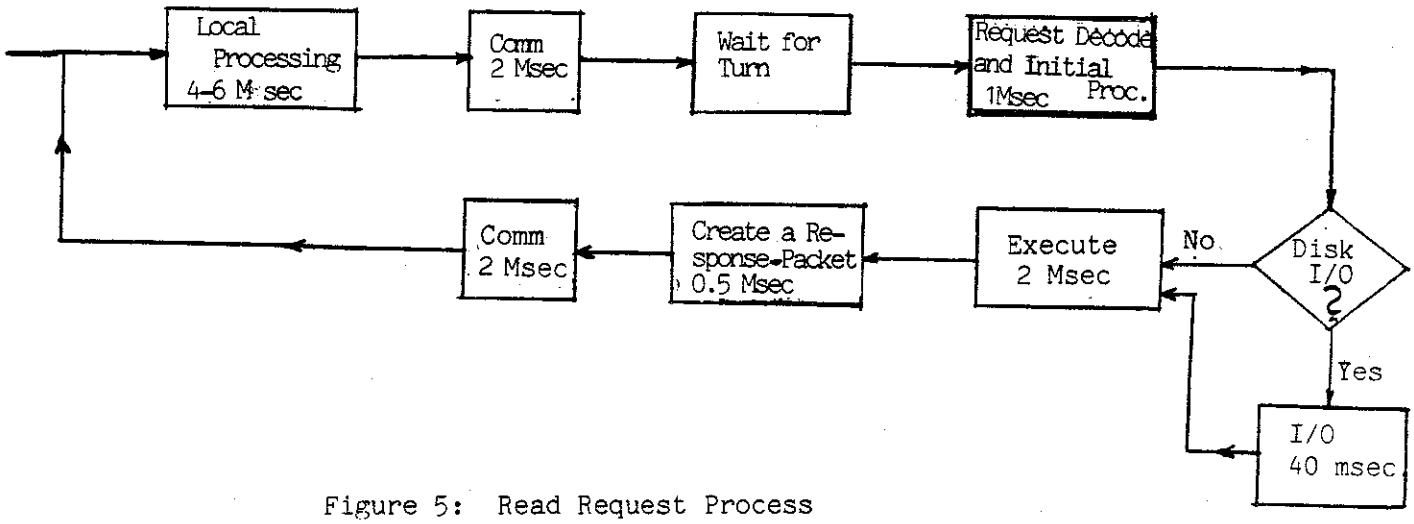

Figure 5: Read Request Process

decoded: if a disk I/O is required to answer the request, it is queued for the disk, otherwise, it is processed immediately. A response packet is then generated by the file- server (containing either the requested data or some other indication, and is then sent from the file server back via the communication line to the requesting workstation which spends some more local processing time to decode the response before preparing the next request. Times shown in Figure 4 and 5 are empirically derived.

As benchmark experiment results came available, the external model was refined to reflect these results in a calibration process, termed "fine-tuning". The satisfactory degree of convergence of the simulation program results with the actual experimental results provided the needed confidence in its validity. In typical practice, data points whose measure of convergence is within a range of 10-15 percent are considered satisfactory. Appendix C presents a SIMSCRIPT program of the external model and an output sample.

ANALYSIS OF EXPERIMENTAL AND SIMULATION RESULTS

The following data were obtained from the benchmarking experiments and their corresponding simulation program results:

1. Average response time (msec.) per request.

2. CPU utilization.

3. I/O pending - Number of records waiting in the disk queue.

4. Request Response Rate - The number of requests (per second) answered by the server.

Of the above parameters, the first is the most representative "user-oriented" performance measure and is illustrated herein. Figures 6-8 show graphic comparisons of measured average response times to those predicted by the model for the "star" configurations (Ogden and Tysons Corner).

Another interesting comparison is provided by Figure 9. This presents the comparison of model-derived average response times showing the effect of "man-in-the-loop". This was achieved by the insertion of 5- and 10-second time delays into the exterior loop of the process Burst Sequence, illustrated in Figure 4, which imparted a "human-dynamic" delay time into each inter-burst sequence. What is shown in the resulting data is that the usual "flat/linear" pattern of response time is obtained but that the transition point between flat and linear is translated out on the x-axis by an order of magnitude. This can be analytically

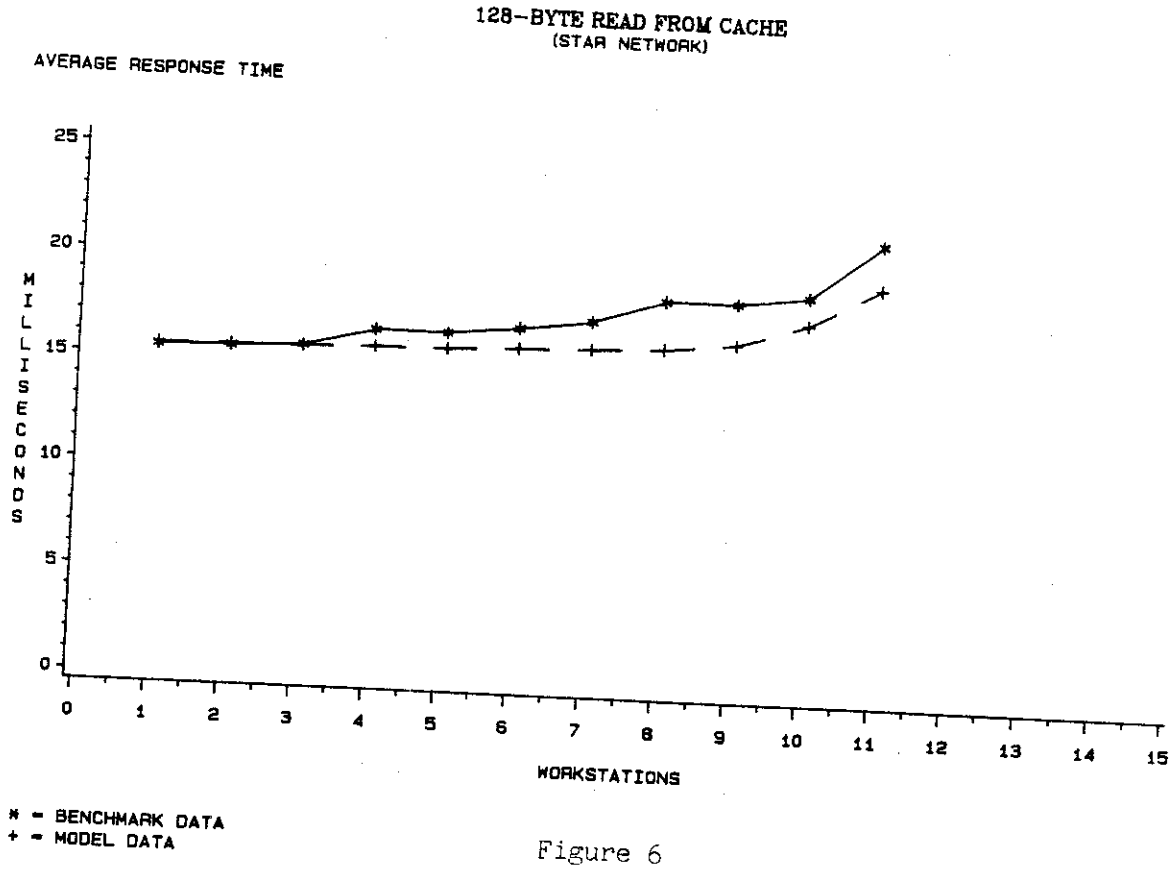

Figure 6

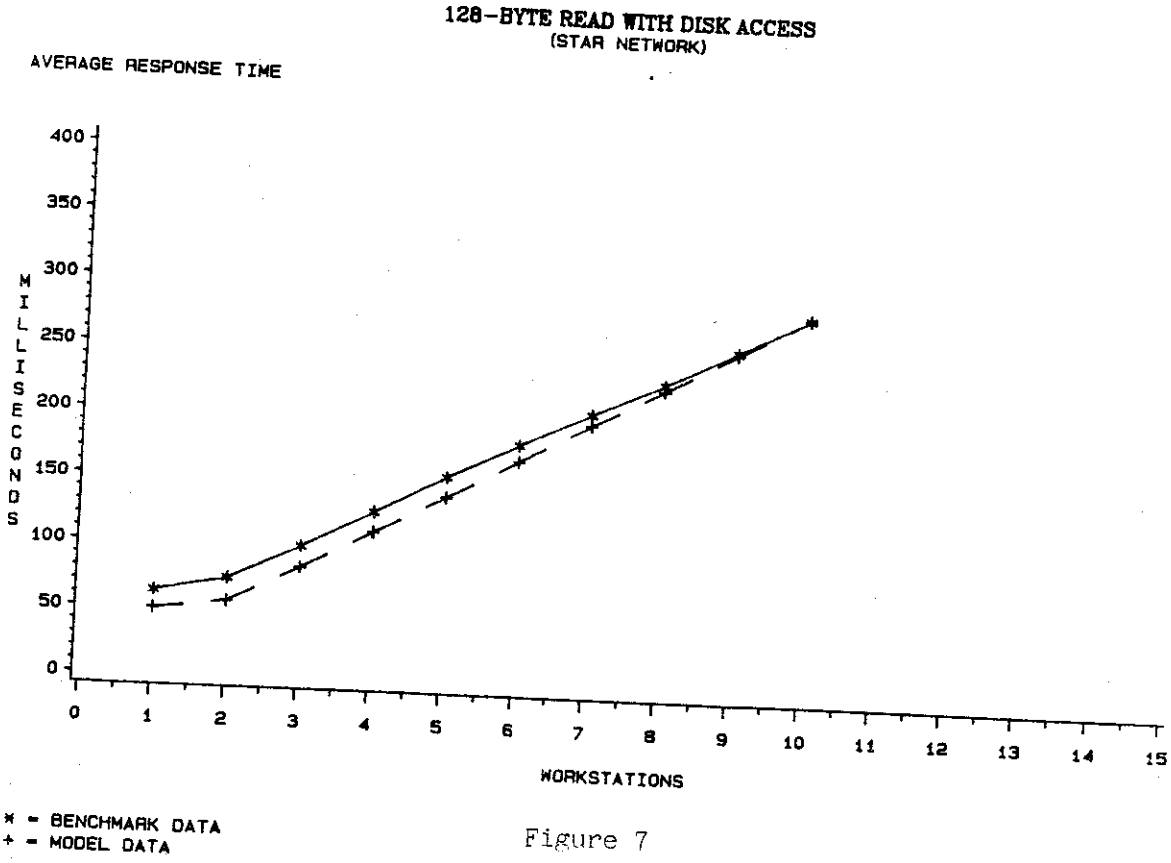

Figure 7

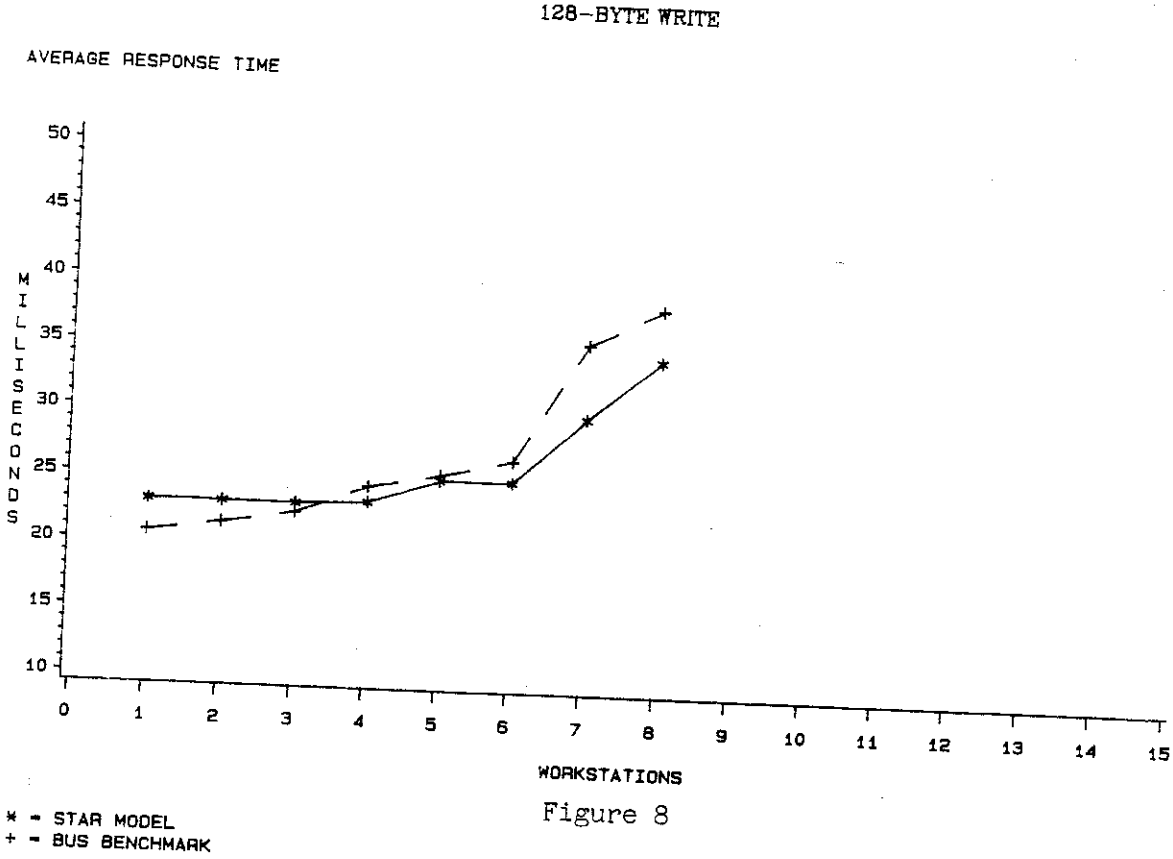

Figure 8

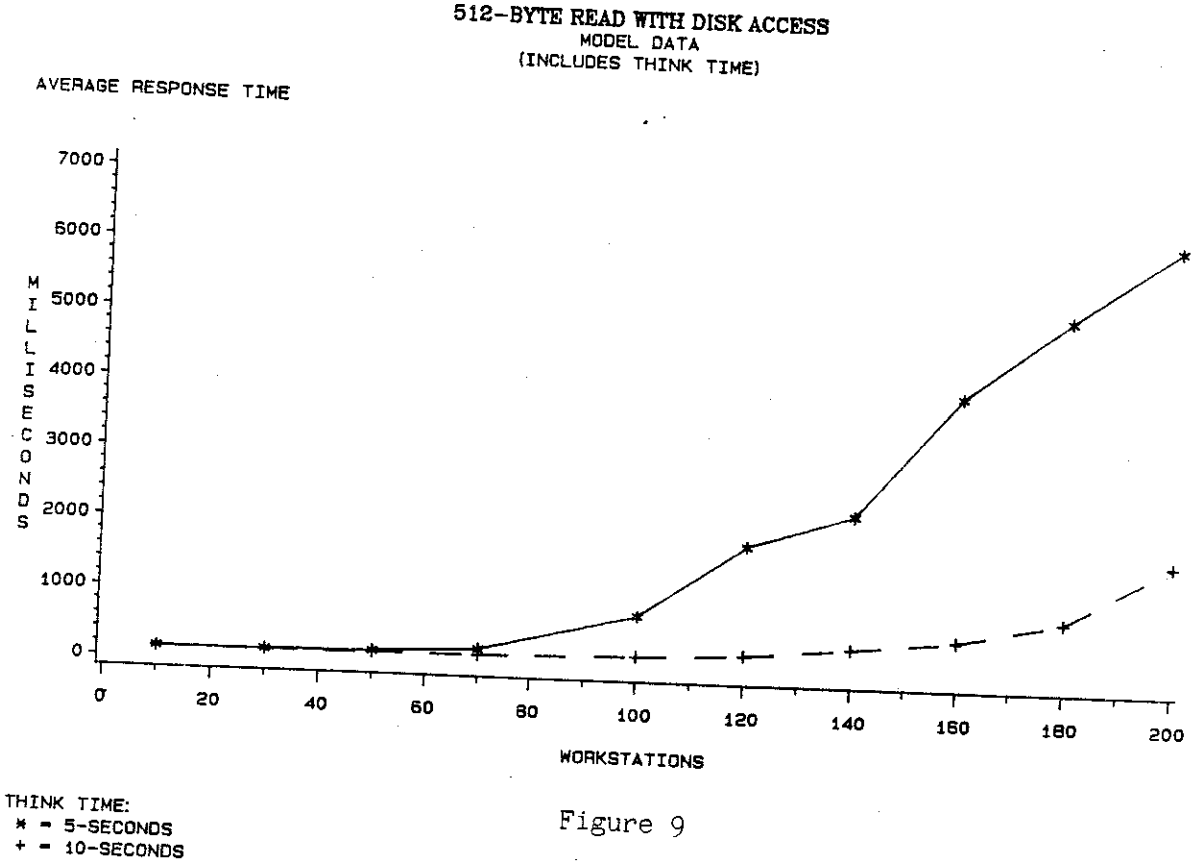

Figure 9

explained by considering that the interburst delay controls the frequency of requests received from a workstation, and that as the delay increases, the frequency of received requests decreases, diminishing the load experienced by the server. A decreased load would result in decreased waiting time, therefore, more workstations could be accomodated. Thus data supports this analysis by showing by comparison of transition points, that more manned stations can be accomodated, and also that the model is sensitive to a variation in the number of stations. One can therefore derive the conclusion that the model is general enough to represent either manned or unmanned configurations. It should also be noted that verification of these predicted results could not be accomplished due to the inability to provide a large (to 150 +) manned workstation environment.

The benchmark data obtained from running the HCI system were not integrated into the analysis of the data obtained elsewhere because they could not be directly compared with data from the simulation model. This was unfortunate because the HCI system provided observation with a 20-workstation network. The reason for the incomparability was that the HCI installation had architectural differences from the system represented in the model: it had a bus architecture and contained a polyglot assortment of microcomputers running at different internal speeds, using different versions of MS-DOS. Even so, this system exhibited internally-consistent response-time profiles so similar to those of the star-oriented system that the implication may be drawn that the current simulation model, with some incremental changes, might be tuned or modified to represent bus, or other, architectures. Figure 10 shows a comparison between reads of different sizes measured on the "bus" system; Figure 11 compares a "bus" measured read with a "star" modeled read. This line of investigation would be an appropriate subject for further research.

From the data presented in the figures, we can identify two performance regions: 1) where response time does not increase with the number of terminals sending saturation level requests (both observed and predicted), and 2) where it increases almost linearly with the number of terminals. The first region, termed the "flat region", is where the system can handle the load without a noticeable increase in response time. The size of this region depends on the type of operation: with requests of small record sizes that do not require disk access, the file server's CPU is the first resource to create a bottleneck in the system.

However, since the processing time involved in such requests is relatively small, the flat region may span as many as 10-13 terminals, all loading the server at the maximum request rate that they can generate.

READ FROM MAIN MEMORY (BUS NETWORK)
(MEASURED)

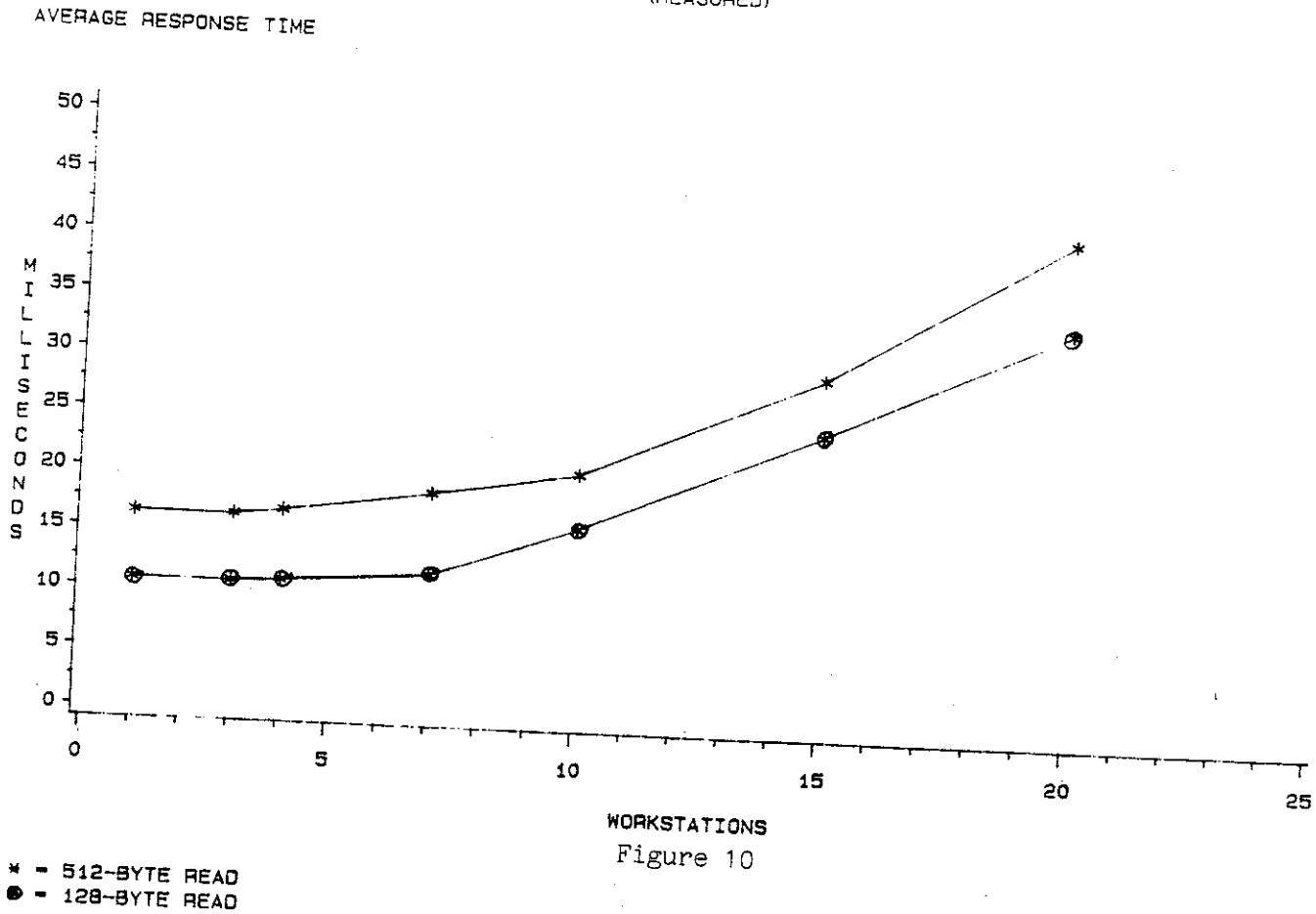

Figure 10

128 BYTE READ FROM MAIN MEMORY
(Bus Benchmark vs Star Model)

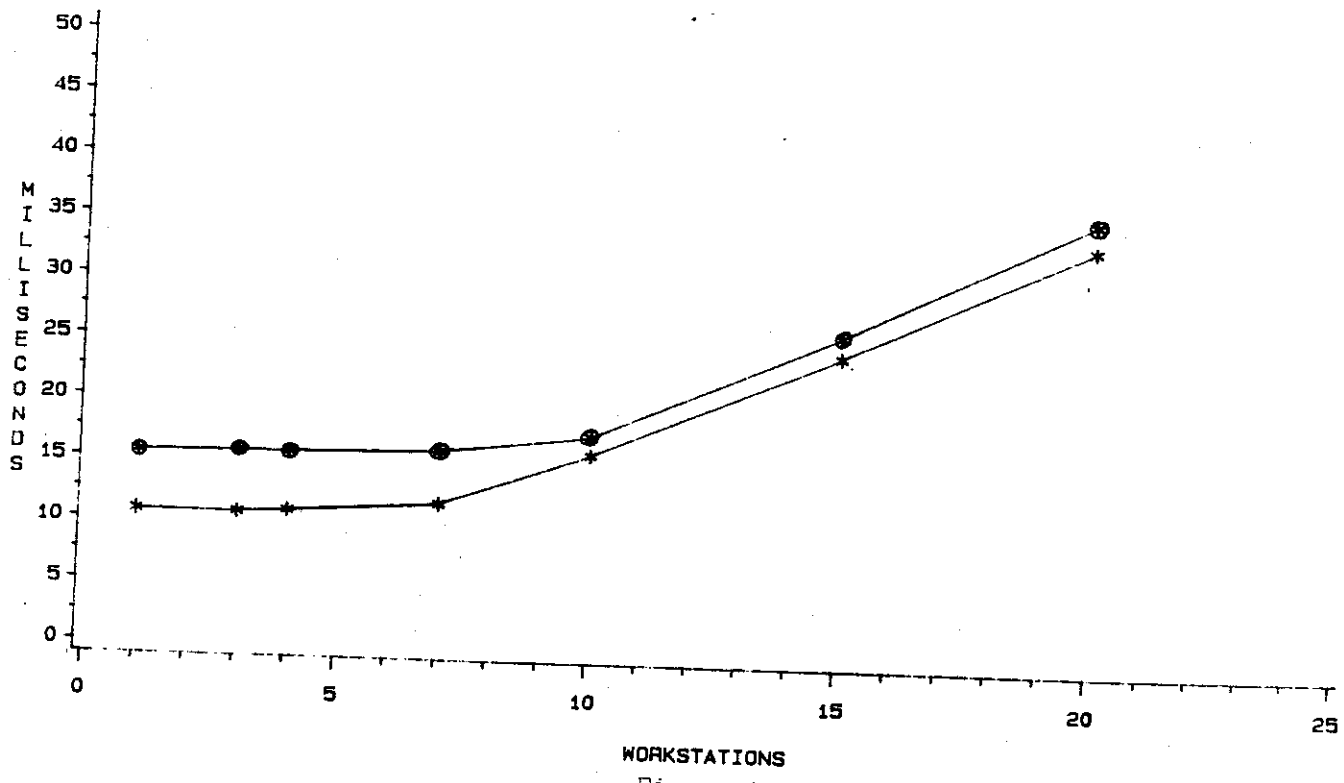

Figure 11

The second performance response region that can be observed from the data as more terminals are added to the experiment, is the "linear region". In this region the system appears to have reached some bottleneck, and works close to its full capacity for this operation: the result is an increase in response time where the corresponding request response-rate and CPU utilization are rising slower and then level off when reaching their capacity limit.

A phenomenon that may be unintuitive at first is the linear increase in response time. An exponential increase would have been less surprising, but this phenomenon can be explained by the fact that this system is not an open, Poisson-driven system. After a station has sent its current request, it is inhibited from sending the next request until a reply to the previous one has been received. This "line-blocking" mechanism is implemented both by the local shell (which, after sending a request packet waits for an event of a reply arrival, or retransmits the request after some timeout expired) and by the NetWare operating system which will discard duplicate or multiple requests from stations which are being currently served. Line blocking ensures that the request arrival rate can never exceed the processing rate, and as a result, imparts a "graceful degradation" - response time increases at a fixed rate - as more terminals are added. Another way of explaining this phenomenon involves the realization that the server input queue can never exceed the number of communication line buffers.

By dividing the observed CPU utilization by the request response rate, an empirical formula was constructed to estimate the required processing time for read requests that do not require disk access. It is a linear equation of the form:

$$T = Ax + b,$$

where T is processing time in milliseconds, x is the record size specified in the request, A is a size coefficient, and b is an overhead factor. Using results obtained for 128 and 512 bytes in the Tyson's experiment set, the A and B factor were obtained and the CPU processing time was found to be:

$$T(msec) = 0.00166*X + 1.597 \quad (X \text{ in bytes}).$$

This formula obtains the maximum processing capacity of the server: for requests with 128-byte record size, capacity is on the order of 550 requests per second. For requests with 512-byte record size, capacity is on the order of 400 requests per second.

A similar analysis was made for requests that involve disk access. In this case more CPU time is required for extra code that needs to be executed plus the time it takes to transfer the data from main memory to the disk channel. By changing the cache block size from 512 bytes to 4096, and repeating the experiment, two sets of results were obtained that, though different, represent the same operation. The difference is due to the fact that in the first set records of 512 bytes were transferred across the disk channel, while on the second, 4096-byte records are transferred. Since disk channel capacity is fixed - in bytes per second - its capacity in records-per-second varies from 35 to 24. This information, plus the CPU utilization recorded during the experiment, led to a second empirical formula that predicts the disk transfer time for a record of size Y to be:

$$T2(msec) = 0.00190*Y + 7.6 \text{ (Y in bytes)},$$

where the coefficient represents the variable size part and the constant represents the extra code involved in performing the disk access. Looking at the disk as a subsystem, a third empirical formula was derived which connects the disk read/write operation to the access time and the transfer rate:

$$T3(msec) = 0.00360*Y + 27 \text{ (Y in bytes)},$$

where the constant part represents the access time.

The above results leads one to view the system as made up of components which are strongly coupled, but which run at different speeds and are sources of different bottlenecks. The disk system is the slowest component; communication the fastest; with CPU-bound processing in the middle. However, because of the line-blocking mechanism, service time from these components stays linear even when their maximum capacity is reached. Figure 12, depicts this idea by showing hypothetical response time in relation to the number of terminals involved in the experiment. Since the service-time curves are not identical for the length of the flat and linear regions for the different subsystems, response time will be affected at a particular number of terminals by one subsystem, and as the number of terminals increase, by another.

This hypothetical portrayal of system behavior should be verified by further experiment and observation.

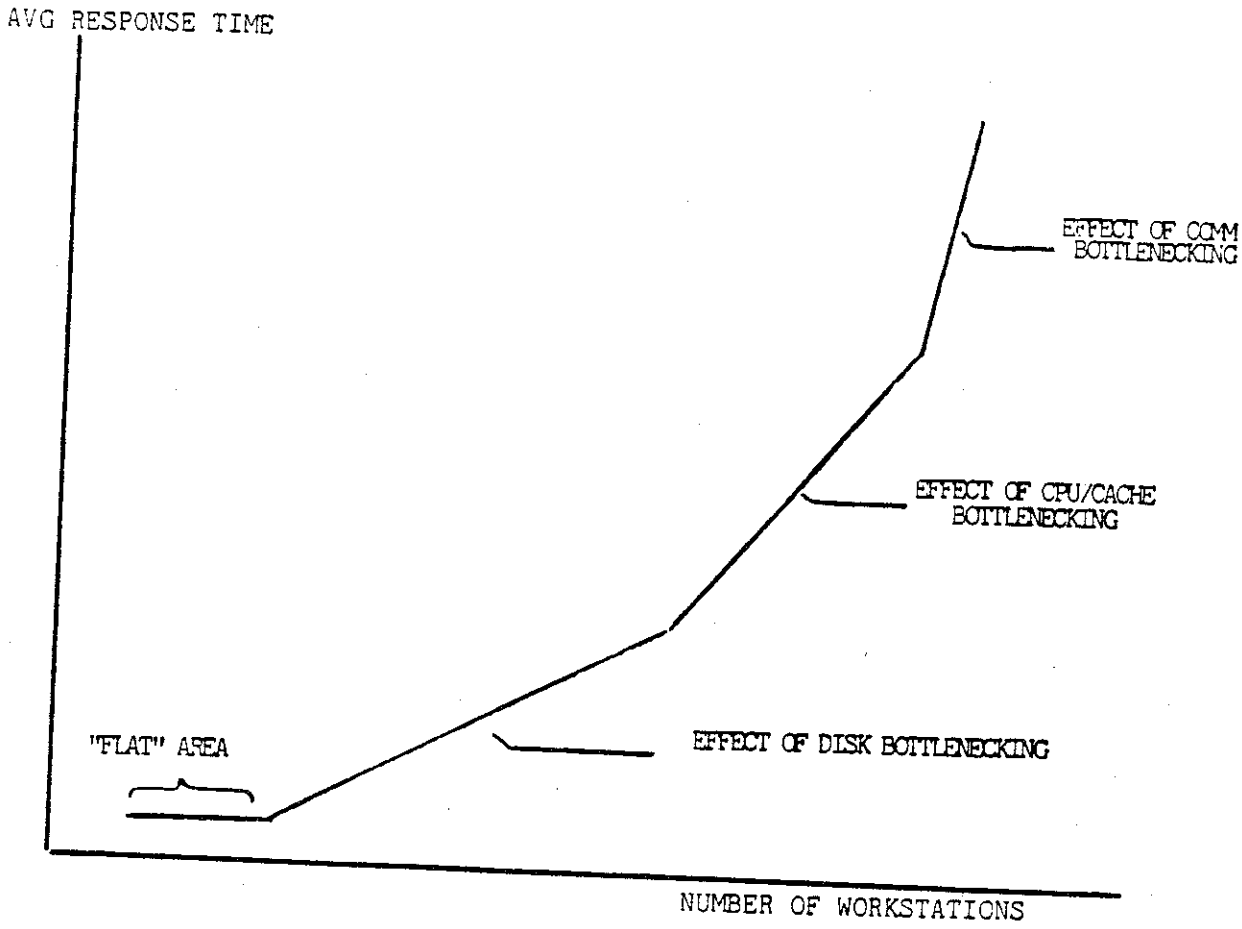

FIGURE 12: Subsystem Effect on System Response Time (Hypothesis)

SIMSCRIPT PROGRAM DESCRIPTION

General

The structure of the program (Appendix B) reflects the modular structure of the model. The program is divided into two functional parts. The first is an auxiliary part that defines the structure of the simulation program, and includes the PREAMBLE, MAIN, STAT.GEN process, INPUT.PARAM subroutine and CUT.SIMUL process. These in effect, are subroutines which initialize the model and generate the output report. The second part contains a description of active objects (e.g., requests) in the system and their interaction. It consists of the STATION, SERVER and DISK processes. These program modules are those that actually execute the simulation process. The following is a detailed description of each module in the SIMSCRIPT program.

PREAMBLE

The PREAMBLE defines the processes and the temporary entities in the model and their means of interaction through the use of global variables and pointers. Process STATION is a module of reentrant code and a copy of the data structure which must be created for each instance of a request from a station. A station must have, therefore, a name (an address to the location of this data-structure, and a RESP.BUFFER which is a pointer to a buffer that contains the responses coming back from the server. The server has a STATE, which is a means to allow synchronization between processes. Unlike the real system, where stations and server run concurrently and independently, processes in a discrete event simulation program are executed one at a time until forced to relinquish the CPU, at which time they may be suspended awaiting the occurrence of some event. Thus, the fact that they are currently not actively running does not mean that they are not "active". State-variables are used to define their activity status and synchronize the active processes. The SERVER process has a unique number (for future multi-server configurations), a DISK.RESP to store responses from the disk, and a REQU.BUFFER to store requests coming from the stations.

Requests and responses are defined as temporary entities in the model just like in the real system. They are created and destroyed by the stations and the server. Each has a unique serial number and other fields that carry information for statistical purposes, generation time and completion time, and for operational purposes, source and destination nodes, SIZE and MODE (SIZE is the request size in bytes and MODE is current

read/write status). The "define to mean" portion of the PREAMBLE allows the expression of time units in milliseconds and seconds; status as BUSY/IDLE; and mode as READ/WRITE. The preamble also defines what statistics should be gathered during the simulation, turnaround time for requests, server utilization, server's input buffer size, and disk input buffer size.

MAIN

The MAIN routine "boots up" the model by reading in the input parameters. It will then invoke the STAT.GEN routine which will create the structure of the modeled network and pass control to the "run-time" simulation controller. The MAIN routine allows repetition of the process with new sets of input data through the use of FOR loop.

Subroutine INPUT.PARAM

This subroutine reads in the input parameters from the input file specified in the JCL commands. It consists of 4 READ statements, therefore there will be 4 lines in the input file for each set of input parameters. It will then print out the given set of input parameters. The following is a description of each parameter in the set:

1. HIT-RATIO - This is an integer number ranging between 1 to 100 which represents the percentage of disk accesses from total number of requests.

2. NUMBER OF STATIONS - As the name implies, it defines the number of stations that will be active in this simulation run. This is an integer number equal to or greater than 1.

3. REQUEST RECORD SIZE - Defines how many bytes will be stored or retrieved in every request (Integer value).

4. CACHE BLOCK SIZE - This is an internal server parameter which defines the size of data block transferred from main memory to disk and vice versa (real number in Kbytes, e.g., 1.0 represents a 1-Kbyte cache block).

5. INTER BURST TIME - This value, in milliseconds, defines the time spent on local processing and "think time" between consecutive bursts of requests (input as a real number).

6. INTER REQUEST TIME - In milliseconds, represents some local processing time between successive request within a burst (input as a real number). Usually short comparing to the inter burst time.

7. BURST LENGTH - Defines the number of request per burst (integer).

8. CACHE SIZE - Size of server's main memory allocated for caching, defined as an integer number of cache blocks (see above).

9. TOTAL NUMBER OF REQUESTS - Defines the total number of requests to be submitted in this simulation run by all stations combined. If mostly concerned with bursts rather than individual requests, multiply total number of desired bursts by the burst size to determine the value of this parameter (integer).

10. READ/WRITE MIX - Defines the ratio of read requests to write requests in the simulation. The integer number, submitted as an input (1 to 100), represents the percentage of read requests.

Parameters 1,2,3 should be entered in the first line, 4,5,6 second, 7,8,9 third, and 10 in the fourth line.

Process STATION

This routine describes the activities performed by a work-station attached to the network. Since the simulation program was intended to be verified by the benchmarking program and vice versa, this process is built around the same process as the benchmarking program. The station generates requests, defines the parameters associated with each request, sends requests to the file server over the channel and waits for responses from the server. Requests are grouped into bursts and there may be delay between bursts and between requests. With each request there is some local processing to prepare the request and decode the response coming back from the server. Some input parameters, such as a request's MODE or TOTAL NUMBER of REQUESTS, directly affect internal variables int his routine, while some like local processing or communication factors are not accessed by the input file, and need to be directly modified should this be required (e.g. when a higher rate communication line is to be used).

The station process interacts with the server process through the use of the REQU.BUFFER and the RESP.BUFFER linked lists, and the STATE variable which indicates the status of the server.

Process SERVER

The SERVER process is the main, most complex routine in this program and effects communicating with stations, controlling the disk and carrying out execution of requests submitted by the workstations. NEW.REQUEST and NEW.RESPONSE are pointers to the

data structures that define requests and responses in their respective sets, REQU.BUFFER, and RESP.BUFFER. MY.DISK is used as a pointer to the DISK process that is used by this server (allows multiple server systems). The DISK process is activated by the SERVER process and they are linked to each other through the use of the DISK.BUFFER (stores requests from the server), and the DISK.RESP (stores responses from the disk). The STATE variable is used t synchronize the server with the other workstations on the network. The STATUS variable is used to convey the status of the disk process and to synchronize it with the server process. The SERVER`s basic cycle starts with checking the REQU.BUFFER, removing the first NEW.REQUEST that is stored there and then begins processing this request.

Write Request Processing

In the case of a write request, a random drawing is made if to determine whether the particular record to be modified is in cache or not (using the HIT.RATIO input variable). If in cache, another test is made to assess whether it was recently modified (but not cleaned yet). If "clean", some internal execution time is spent, and the request is filed in the DISK.BUFF. Subroutine EXEC is called then to generate the response for this request, it will acknowledge to the workstation that the request has been accepted. If that the "specified" record is not in cache, another drawing is made to pick a cache block at random (The real system applies a "Least Recently Used" algorithm). If the chosen block is clean, it proceeds as before, otherwise this block must be flushed to disk before it can be rewritten. This is done by changing the MODE variable of this request to QWRITE.CLEAN, which will signal the server that another disk-write is required when it checks the response from the disk.

Read Request Processing

If the request is a READ, a draw will be made to determine if the specified record is in cache or not, it is. EXEC is called into action to prepare the response packet and send it back. If the record is not in cache, a block is chosen at random. If the chosen record is dirty it is cleaned first in a process similar to the WRITE case. The request is then placed at the DISK.BUFF to cause data-retrieval from the disk. The above process repeats itself until the REQU.BUFFER is empty. The next loop deals with responses that come back from the disk process.

Disk Response Processing

While the DISK.RESP buffer is not empty it will remove the first item in the set (a semi-processed request). After some execution time involved in transferring the data to main memory, the response is checked for its MODE. If it is a READ, EXEC is called to prepare and send the response. IF MODE is QREAD.CLEAN, it needs to pass one more time through the DISK process, to "fetch" the requested data, thus it is put back in the DISK.BUFF. If MODE is QWRITE.CLEAN, it also goes back to disk processing, but a response is created and sent immediately. After all disk responses are processed, the server changes its STATE to IDLE, and suspends itself until new requests are sent from the stations, or new responses arrive from the disk.

Subroutine EXEC:

This module of the program generates the responses, assigns values to their parameters, and sends them over the communication line to their destination. The processing time involved with each request is defined by the PRTIME variable calculated for each request as a function of its size plus some overhead time. COMTIME is a variable that defines the communication time for each request, also a function of its size and some overhead time. The coefficients for the calculation of these variables were derived empirically from the benchmarking results.

This subroutine also has some garbage-cleaning duty to destroy the requests after their corresponding responses - all but those with WRITE mode, which are destroyed by the DISK process - have been sent back.

Process DISK

This program module simulates the functions performed by the disk. Basically, it is made of a loop that removes requests from the DISK.BUFF, using DE.REQUEST as a pointer to the data structure. It will then spend some time in locating and retrieving/updating the requested record. The combined seek and transfer time is calculated for each request based on the record size as defined by the BLOCK.SIZE input parameter, plus some overhead time. It will then place the response for that request back in the DISK.RESP buffer unless its MODE is QWRITE, a case in which it will simply destroy the request (a response for WRITES is sent before the information is actually written to the disk).

FUTURE RESEARCH

Having established the status of the project reported herein and drawing from the results and conclusions, it is possible to establish various routes for further research, with objectives of improving the depth of knowledge, increasing the breadth, or improving useability of the technique. Figure 13 serves as a map to those routes.

Extend NetWare Functions

Since only the read and write processes are incorporated in the model, the addition of other NetWare activities such as file locking, file open and close, etc., would enable the expansion of the model's utility. This means that more complex user-generated streams could be simulated, thus imparting more realism to the model's application.

Increase Model Detail

The NetWare operating system is represented very basically in the "external" model. By adding modules, representing such functions as cache utilization and disk track queuing, a more detailed representation of the system would be obtained. This would enable the prediction of performance sensitivity to secondary effects with the "external" model.

External Configuration

In practice, several file servers can be inter-connected, thus imparting a very large distributed file capability to the system. One server calls upon another to locate requested files; multiple references of this sort can be accommodated. The "external" model could be extended to represent such multiples enabling the prediction of performance tradeoffs with data organization, routing, etc.

Extend Benchmarking

Validation of the current model has been limited by the size of benchmarking facilities available. Two methods to increase the benchmarking capabilities are to: 1) have a very large network available, preferably with homogeneous workstations, and availability of various communication packages; 2) extend the research to investigation of a terminal emulator, which invokes the use of one machine to send requests into the network as if there were many.

DIRECTIONS FOR GROWTH OF PC-LAN SIMULATION RESEARCH

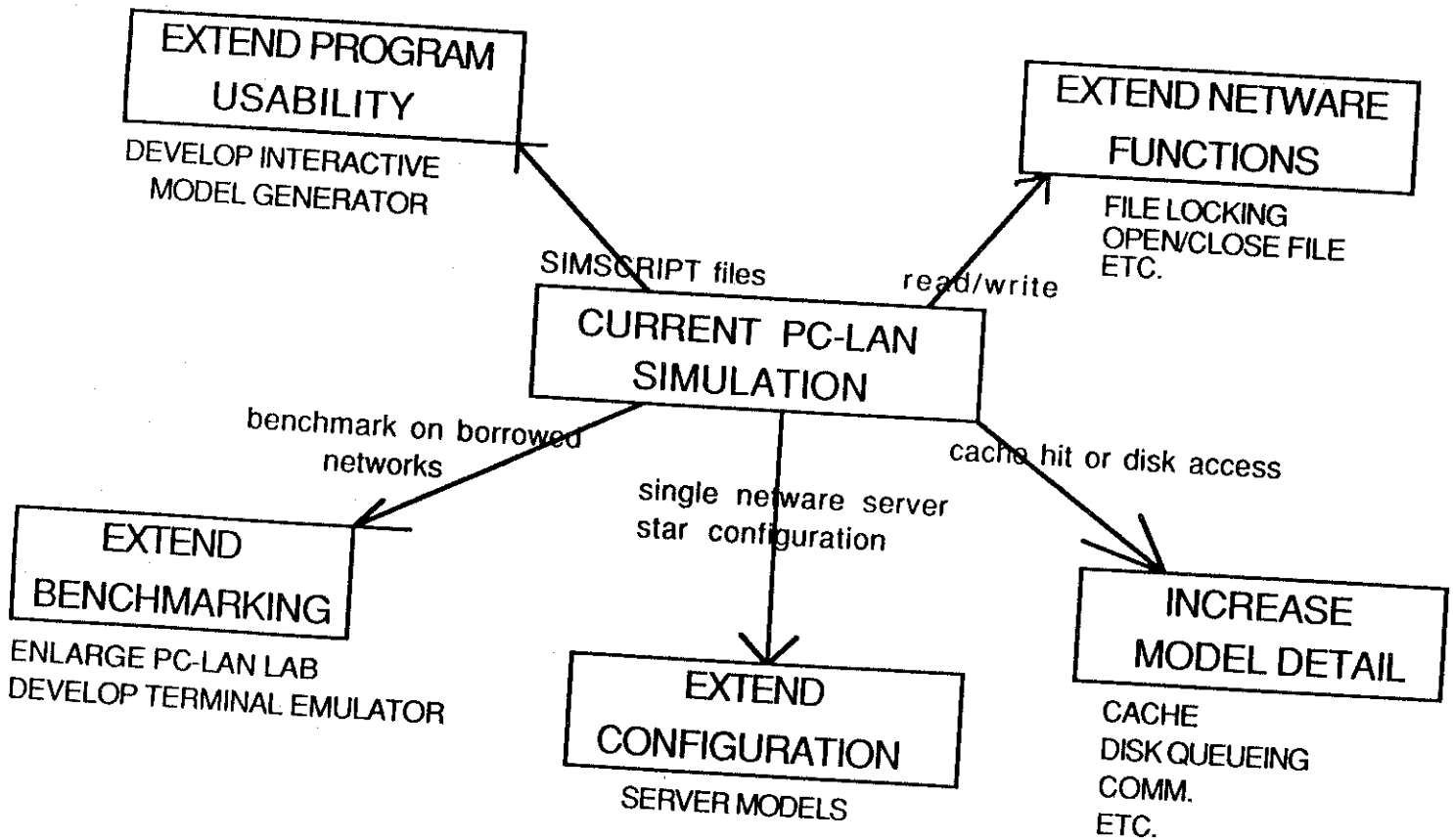

Figure 13

Extend Program Useability

To enable managers and personnel not necessarily experienced in simulation modeling to configure systems readily, a front-end processor which works interactively to generate a model might be developed. The technology appears, in general, to be available, although interactive processors for this specific application have apparently not been developed and/or widely employed.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the significant contributions of the Applied Research staff at IRS, Stuart Milner, Chief, and his assistants, Rick Heroux and Aaron Benett. Not only did Dr. Milner make contributions to the planning and implementation of the research, he arranged for all of the Novell contacts and the benchmarking sessions; then he and his group contributed personally to the benchmark running, often working long and late hours "on location".

Also due thanks is Novell, Inc., who graciously gave us access to corporate technical staff, especially Kyle Powell, who gave generously of his heavily scheduled time to first explain how the system worked, and then to critique the model. Also, Mark Bennett of Novell at Tyson's Corner and his staff made their LAN training facility available for extended periods of time so that we could complete several benchmarking experiments. Reid Clark, of Novell, secured for us a complete NetWare software package, so that we could implement a server application on our laboratory LAN at Tech, which was purchased with funds under this grant. Acknowledgment is also given to Novell, Inc., for the use of Figures 1-3.

Finally, acknowledgement is due to Michael Van Patten, of HCI, and his staff for giving us our only opportunity to access a large LAN.

APPENDIX A BENCHMARK PROGRAM (BASICa)

```
5 CLS
6 PRINT "A STATISTICAL OBSERVATION PROGRAM"
7 PRINT "-----------------------------------"
8 PRINT
10 PRINT "To start execution, put in the following parameters:"
11 INPUT "Enter file name (file will be destroyed during exec)";FLNAME$
12 IF FLNAME$ = "def" GOTO 700
13 INPUT "Enter max. file-size" ; FLSIZE
15 INPUT "Enter record length (1 to 1024 bytes) "; LL
16 IF LL > 1024 THEN GOTO 200
20 INPUT "Enter number of records to be read or written ";NUM
21 INPUT "Type R for READ or W for WRITE" ; OPER$
22 INPUT "Type S for SEQUENTIAL or R for RANDOM file access"; RAND$
23 IF OPER$ <> "R" AND OPER$ <> "W" THEN GOTO 200
24 IF RAND$ <> "S" AND RAND$ <> "R" THEN GOTO 200
25 OPEN "results" FOR APPEND AS #2
26 PRINT #2, "RECORD LENGTH = ";LL;", NUMBER OF RECORDS = "; NUM
27 SUM = 0
28 PRINT "The test will be repeated 100 times, then stats will be computed"
29 INPUT "COMMENTS ";BOM$
30 IF OPER$ = "R" THEN PRINT #2, "Reading" ELSE PRINT #2, "Writing"
40 IF RAND$ = "S" THEN PRINT #2, "Sequential" ELSE PRINT #2, "Random"
41 PRINT #2, BOM$
45 RANGE = INT(FLSIZE/LL) - 1
50 RANDOMIZE TIMER
51 PRINT "the time now is "; TIME$
52 INPUT "Enter time to start execution"; T$
53 V$ = MID$(T$,4,2)
54 TT$ = MID$(TIME$,4,2)
55 LOCATE 18,35
56 PRINT TIME$
58 IF V$ <> TT$ GOTO 54
59 TSTART = TIMER
60 FOR I = 1 TO 100
70 OPEN FLNAME$ AS #1 LEN = LL
71 IF OPER$ = "R" AND RAND$ = "S" THEN GOSUB 300
72 IF OPER$ = "W" AND RAND$ = "S" THEN GOSUB 400
73 IF OPER$ = "R" AND RAND$ = "R" THEN GOSUB 500
74 IF OPER$ = "W" AND RAND$ = "R" THEN GOSUB 600
97 IF OPER$ = "R" THEN PRINT I; "  Reading operation took "; E ; " miliseconds"
98 IF OPER$ = "W" THEN PRINT I;"  Writing operation took "; E ; " miliseconds"
100 CLOSE #1
105 SUM=SUM + E
120 NEXT
125 TSTOP = TIMER
140 PRINT #2,"AVERAGE PER REPETITION = "; SUM/100; " MILLISECONDS"
145 PRINT #2,"AVERAGE PER BYTE = "; SUM/(100 * NUM * LL); " MILLISECONDS"
146 PRINT #2,"TOTAL EXPERIMENT TIME =";TSTOP-TSTART ; "SECONDS"
147 PRINT #2,"TOTAL NUMBER OF REQUESTS = ";NUM*100
148 PRINT #2,
149 PRINT #2,
150 END
200 PRINT "Bad input parameters, rerun program"
210 STOP
300 D = TIMER
310 FOR J = 1 TO NUM
320 GET 1
330 NEXT
340 E = (TIMER - D) * 1000
350 RETURN
```

```
400 D = TIMER
410 FOR J = 1 TO NUM
420 PUT 1
430 NEXT
440 E = (TIMER - D) * 1000
450 RETURN
500 D = TIMER
510 FOR J = 1 TO NUM
520 GET 1, INT(RND * RANGE)+1
530 NEXT
540 E = (TIMER - D) * 1000
550 RETURN
600 D = TIMER
610 FOR J = 1 TO NUM
620 PUT 1, INT(RND * RANGE)+1
630 NEXT
640 E = (TIMER - D) * 1000
650 RETURN
700 FLNAME$ = "f:junk2.gps"
710 FLSIZE = 60000!
720 LL = 1
730 NUM = 100
740 OPER$ = "R"
750 RAND$ = "S"
760 GOTO 25
```

RECORD LENGTH = 128 , NUMBER OF RECORDS = 100
Reading
Sequential
file in cache 1 station
AVERAGE PER REPETITION = 1541.055 MILLISECONDS
AVERAGE PER BYTE = .1203949 MILLISECONDS
TOTAL EXPERIMENT TIME = 176.4805 SECONDS
TOTAL NUMBER OF REQUESTS = 10000

RECORD LENGTH = 128 , NUMBER OF RECORDS = 100
Reading
Sequential
2 stations
AVERAGE PER REPETITION = 1548.047 MILLISECONDS
AVERAGE PER BYTE = .1209412 MILLISECONDS
TOTAL EXPERIMENT TIME = 177.1914 SECONDS
TOTAL NUMBER OF REQUESTS = 10000

RECORD LENGTH = 128 , NUMBER OF RECORDS = 100
Reading
Sequential
3 stations
AVERAGE PER REPETITION = 1556.094 MILLISECONDS
AVERAGE PER BYTE = .1215698 MILLISECONDS
TOTAL EXPERIMENT TIME = 178.1797 SECONDS
TOTAL NUMBER OF REQUESTS = 10000

RECORD LENGTH = 128 , NUMBER OF RECORDS = 100
Reading
Sequential
4 stations
AVERAGE PER REPETITION = 1603.401 MILLISECONDS
AVERAGE PER BYTE = .1252657 MILLISECONDS
TOTAL EXPERIMENT TIME = 183.83 SECONDS
TOTAL NUMBER OF REQUESTS = 10000

RECORD LENGTH = 128 , NUMBER OF RECORDS = 100
Reading
Sequential
5 stations
AVERAGE PER REPETITION = 1609.897 MILLISECONDS
AVERAGE PER BYTE = .1257732 MILLISECONDS
TOTAL EXPERIMENT TIME = 184.1599 SECONDS
TOTAL NUMBER OF REQUESTS = 10000

RECORD LENGTH = 128 , NUMBER OF RECORDS = 100
Reading
Sequential
6 stations
AVERAGE PER REPETITION = 1637.799 MILLISECONDS
AVERAGE PER BYTE = .1279533 MILLISECONDS
TOTAL EXPERIMENT TIME = 186.86 SECONDS
TOTAL NUMBER OF REQUESTS = 10000

```
;  GPSS/PC program file NOVEL1.  (V 1.1, # 30640)  07-18-1986 15:37:33
1 EXPON      FUNCTION      RN8,C24
0.,0/0.1,0.104/0.2,0.222/0.3,0.355/0.4,0.509/0.5,0.69/0.6,0.915/
0.7,1.2/0.75,1.38/0.8,1.6/0.84,1.83/0.88,2.12/0.9,2.3/0.92,2.52/
0.94,2.81/0.95,2.99/0.96,3.2/0.97,3.5/0.98,3.9/0.99,4.6/0.995,5.3/
0.998,6.2/0.999,7.0/0.9997,8.0
20           INITIAL      X$REQ,0
25           INITIAL      X$CNTR,0
30           INITIAL      X$POINT,0              ;USED IN THE CACHE PROCESS
35 FLAG      MATRIX       ,1,20
40           INITIAL      MX$FLAG(1,20),0
50 CACHE     MATRIX       ,2,20
55           INITIAL      MX$CACHE(1,20),0       ;20 BLOCKS OF CACHE MEMORY
60           INITIAL      X$DIRBT,0
65           INITIAL      X$PIC,0                ;DIRTY BLOCKS COUNTER
                                                 ;AUXILIARY VARIABLE IN THE CACHE P
66 SERV      STORAGE      8
70 MOVE      BVARIABLE    ((S$SERV'LE'7)'AND'(X$REQ'GE'1))
75 ENTR      BVARIABLE    ((S$SERV'LE'7)'AND'(X$POINT'E'P3)'AND'(FV$CPU))
80 FULL      FUNCTION     RN5,D2                 ;CACHE EMPTY 90 PERCENT OF TIME
 .9,0/1,1
85 ONE       FUNCTION     RN3,D2                 ;50% READS 50% WRITES
 .5,0/1,1
90 TWO       FUNCTION     RN2,D2                 ;80% HIT RATIO
 .2,0/1,1
95 THREE     FUNCTION     RN4,D2                 ;5 % OF REQUESTS ARE LOCKED
 .95,0/1,1
100 FOUR     FUNCTION     RN5,C2                 ;UNIFORMLY DISTRIBUTED FROM 1 TO 1
0,1/1,11
105 FIVE     FUNCTION     RN8,C2                 ;40 DISK BLOCKS TO CHOOSE FROM
0,1/1,41
110 RAND     FUNCTION     RN7,C2                 ;PICK 1 OF THE 20 BLOCKS FOR REPLA
0,1/1,21
115 DISTR    TABLE        M1,0,50,100
120 FILES    TABLE        P4,0,1,42
130          GENERATE     10000,FN$EXPON,,,      ;MEAN IS 1 SECOND
135          QUEUE        SYS
140          ASSIGN       1,FN$ONE               ;ASSIGN  TYPE OF REQUEST
145          ASSIGN       4,FN$FIVE              ;ASSIGN  BLOCK NUMBER TO ACCESS
150          ASSIGN       5,FN$THREE             ;ASSIGN  LOCKED/UNLOCKED
155          ASSIGN       3,FN$FOUR              ;ASSIGN  A SOURCE NODE ( FROM 1 TO
160          SAVEVALUE    REQ+,1                 ;INCREMENT REQUEST COUNTER
165          ADVANCE      1                      ;TO WAKE UP THE MUX PROCES
170          QUEUE        POLL
175          TEST E  ,    BV$ENTR,1              ;WAIT FOR A MUX PROCESS TO POINT
180          SAVEVALUE    REQ-,1                 ;DECREMENT REQUEST COUNTER
185          DEPART       POLL
190          MSAVEVALUE   FLAG,1,P3,1            ;SET SOURCE-IN-SERVICE FLAG
195          SEIZE        CPU
200          ENTER        SERV                   ;ASSIGN A SERVER PROCESS
205          ADVANCE      30,10                  ;WAIT 3 MSEC FOR REQUEST DECODING
210          RELEASE      CPU
215          TEST E       P1,0,RITE              ;TEST IF READ/WRITE
220          SEIZE        CPU
225          SAVEVALUE    CNTR,0                 ;RESET CACHE BLOCK COUNTER
```

```
230 LOP     SAVEVALUE   CNTR+,1
234         ASSIGN      7,X$CNTR                    ;TO TRICK THE COMPILER
235         TEST NE     MX$CACHE(1,P7),P4,INC       ;CHECK IF DESIRED FILE IN CACHE

240         ADVANCE     5                           ;THE TIME IT TAKES TO SEARCH

245         TEST E      X$CNTR,30,LOP               ;SEARCHED WHOLE CACHE ?
250         RELEASE     CPU                         ;NO NEED FOR CPU ON I/O OPERATION

255         SEIZE       CHANL
260         ADVANCE     200,50                      ;BRING FILE TO CACHE
                                                    ;WAIT UNTIL READING IS COMPLETE
265 SEC     SAVEVALUE   PIC,FN$RAND                 ;PICK A CACHE BLOCK AT RANDOM

267         ASSIGN      7,X$PIC
270         TEST E      MX$CACHE(2,P7),0,SEC        ;TO TRICK THE COMPILER
275         MSAVEVALUE  CACHE,1,X$PIC,P4            ;SURE IT'S NOT DIRTY
                                                    ;PUT FILE IN THE CACHE BLOCK
280         RELEASE     CHANL                       ;USING A RANDOM REPLACEMENT POLICY

285         SEIZE       CPU
290 INC     ADVANCE     30                          ;TRANSFER FROM CACHE TO MAIN MEM

295         ADVANCE     20                          ;TRANSFER FROM MAIN MEM TO I/O POR

300         RELEASE     CPU
310         TRANSFER    ,TERMNT
315 RITE    SEIZE       CPU
320         SAVEVALUE   CNTR,0
325 LOOP2   SAVEVALUE   CNTR+,1                     ;ESET CACHE BLOCK COUNTER
327         ASSIGN      7,X$CNTR
330         TEST NE     MX$CACHE(1,P7),P4,IMID      ;TRICK IT

335         ADVANCE     5                           ;CHECK IF ALREADY IN CACHE

                                                    ;THE TIME IT TAKES TO SEARCH
340         TEST E      X$CNTR,20,LOOP2             ;ALL BLOCKS CHECKED ?
345         TEST E      X$DIRBT,20,REPL             ;ARE ALL BLOCKS "DIRTY" ?
350         SEIZE       CHANL                       ;IF SO, GOT TO FLUSH CACHE TO DISK

355         SAVEVALUE   CNTR,0

360 LOOP3   SAVEVALUE   CNTR+,1                     ;IMMIDIATELY TO ALLOW THE WRITE RE
365         MSAVEVALUE  CACHE,2,X$CNTR,0            ;RESET THE DIRTY BIT FOR THIS BLOC

370         RELEASE     CPU

375         ADVANCE     200,50                      ;DO NOT TIE CPU WHILE I/O IN PROGR
                                                    ;THE TIME IT TAKES TO WRITE TO DIS
380         SEIZE       CPU
385         ADVANCE     5                           ;NEED TH ECPU AGAIN
                                                    ;SOME TIME FOR THAT BOOK-KEEPING
390         TEST E      X$CNTR,20,LOOP3             ;CONTINUE UNTIL ALL BLOCKS CLEARED
```

```
395             RELEASE     CHANL                       ;FREE THE CHANL WHEN FINISHED
400             SAVEVALUE   DIRBT,0                     ;RESET THE DIRTY BLOCKS COUNTER
405 REPL        ASSIGN      7,FN$RAND                   ;PICK A CACHE BLOCK TO WRITE INTO
410             TEST E      MX$CACHE(2,P7),0,REPL  ;CHECK IF IT'S NOT DIRTY
415             MSAVEVALUE  CACHE,1,P7,P4               ;WRITE TO CACHE
420             MSAVEVALUE  CACHE,2,P7,1                ;MARK THIS BLOCK AS "DIRTY"
425             SAVEVALUE   DIRBT+,1                    ;INCREMENT DIRTY BLOCKS COUNTER
430             ADVANCE     40                          ;THE TIME IT TAKES TO WRITE TO CAC
435             TRANSFER    ,SKIP
440 IMID        ADVANCE     40                          ;WRITE INTO CACHE AND REREAD
445             MSAVEVALUE  CACHE,2,X$CNTR,1            ;MARK THIS BLOCK AS DIRTY
450             SAVEVALUE   DIRBT+,1                    ;INCREMENT DIRTY BLOCKS COUNTER
455 SKIP        RELEASE     CPU
460             TRANSFER    ,TERMNT
465 LOCK        ADVANCE     1000                        ;WAIT 0.1 SEC
470             ASSIGN      5,FN3                       ;CHECK IF UNLOCKED
475             GENERATE    ,,,1,                       ;ONLY ONE TRANSACTION
480             PRIORITY    2                           ;LOWER PRIORITY THEN REQUESTS
485 EXLOP       TEST GE     X$DIRBT,1                   ;ARE THERE DIRTY BLOCKS ?
490             SAVEVALUE   CNTER,0
495             SEIZE       CPU                         ;MUST HAVE THE CPU AND THE I/O CHA
500             SEIZE       CHANL
505 INLOP       SAVEVALUE   CNTER+,1
507             ASSIGN      8,X$CNTER                   ;INCREMENT BLOCK COUNTER
510             TEST E      MX$CACHE(2,P8),1,JMP3  ;IS THIS BLOCK DIRTY ?
515             ADVANCE     200,50                      ;IF SO WRITE IT TO THE DISK
520             MSAVEVALUE  CACHE,2,X$CNTER,0           ;RESET THE DIRTY BIT FOR THIS BLOC
525             SAVEVALUE   DIRBT-,1                    ;DECREMENT DIRTY BLOCKS COUNTER
530 JMP3        TEST E      X$DIRBT,0,INLOP             ;ANY DIRTY BLOCKS LEFT ?
535             RELEASE     CPU
540             RELEASE     CHANL                       ;FREE CPU AND I/O CHANNEL
```

```
545
550 TERMNT  SEIZE       CPU                 ;PASSIVATE UNTIL NEXT ROUND
555         LEAVE       SERV
560         ADVANCE     10                  ;LEAVE THE SERVER
                                            ;SPEND  1 MSEC ON HOUSE-KEEPING
565         RELEASE     CPU
570         DEPART      SYS
575         MSAVEVALUE  FLAG,1,P3,0         ;RESET FLAG TO ALLOW NEXT REQ. FRO

580         TABULATE    DISTR               ;THAT NODE
585         TERMINATE   1
590         GENERATE    ,,,1                ;ONE TRANSACTION
595 REST    SAVEVALUE   POINT,0
600 COT     TEST E      BV$MOVE,1           ;TEST IF POINTER SHOULD BE MOVED

605         SEIZE       CPU                 ;MUX PROCESS ALSO REQUIRES CPU-TIM

610         ADVANCE     5                   ;SOME PROCESSING TIME
620         RELEASE     CPU
625         SAVEVALUE   POINT+,1            ;INCREMENT POINTER
630         TEST NE     X$POINT,11,REST     ;TEST IF END OF CYCLE
633         ASSIGN      6,X$POINT
635         TEST NE     MX$FLAG(1,P6),1,SKIP3   ;ERROR PRONE
640         ADVANCE     5                   ;WAIT FOR REQUEST TO ENTER SYSTEM

645 SKIP3   GATE SF     SERV,COT            ;CHECK IF ALL SERVERS BUSY
650         SAVEVALUE   TEMP,X$POINT        ;IF SO,  STOP THE SCANNING
655         SAVEVALUE   POINT,0             ;AND RESUME WHEN SERVERS ARE AVAIL

660         GATE SNF    SERV
665         SAVEVALUE   POINT,X$TEMP
670         TRANSFER    ,COT
```

IAT = 50 MSEC, MULTIPLE REQUESTS FROM SAME STATION
ARE DISCARDED

START_TIME	END_TIME	BLOCKS	FACILITIES	STORAGES	FREE_MEMORY
0	279085	114	2	1	126464

FACILITY	ENTRIES	UTIL.	AVE._TIME	AVAILABLE	OWNER	PEND	INTER	RETRY	DELAY
CPU	4083	0.508	34.73	1	527	0	0	0	0
CHANL	379	0.271	200.02	1	0	0	0	0	0

QUEUE	MAX	CONT.	ENTRIES	ENTRIES(0)	AVE.CONT.	AVE.TIME	AVE.(-0)	RETRY
SYS	5	1	501	0	0.98	544.54	544.54	0
POLL	4	0	501	25	0.29	162.27	170.79	0

STORAGE	CAP.	REMAIN.	MIN.	MAX.	ENTRIES	AVL.	AVE.C.	UTIL.	RETRY	DELAY
SERV	8	7	0	5	501	1	0.63	0.079	1	0

TABLE DISTR — MEAN 543.56 — STD.DEV. 290.05 — RETRY 0

RANGE			FREQUENCY	CUM. %
50	–	100	1	0.20
100	–	150	14	3.00
150	–	200	28	8.60
200	–	250	18	12.20
250	–	300	15	15.20
300	–	350	35	22.20
350	–	400	40	30.20
400	–	450	67	43.60
450	–	500	58	55.20
500	–	550	42	63.60
550	–	600	22	68.00
600	–	650	19	71.80
650	–	700	27	77.20

TABLE	MEAN	STD. DEV.	RETRY	RANGE		FREQUENCY	CUM. %
				700 -	750	25	82.20
				750 -	800	16	85.40
				800 -	850	11	87.60
				850 -	900	16	90.80
				900 -	950	4	91.60
				950 -	1000	6	92.80
				1000 -	1050	6	94.00
				1050 -	1100	6	95.20
				1100 -	1150	1	95.40
				1150 -	1200	3	96.00
				1200 -	1250	4	96.80
				1250 -	1300	2	97.20
				1300 -	1350	2	97.60
				1350 -	1400	1	97.80
				1400 -	1450	3	98.40
				1450 -	1500	1	98.60
				1550 -	1600	2	99.00
				1600 -	1650	2	99.40
				1650 -	1700	1	99.60
				1800 -	1850	1	99.80
				1950 -	2000	1	100.00
FILES	0.00	0.00	0				

SAVEVALUE	VALUE	RETRY
REQ	+0	1
CNTR	+1	0
POINT	+2	0
DIRBT	+0	1
PIC	+19	0
CNTER	+1	0

MATRIX	RETRY	ROW	COLUMN	VALUE
FLAG	0			
		1	1	+0
		Intermediate values are 0		
		1	6	+1
		Intermediate values are 0		
CACHE	0	1	20	+0
		1	1	+11
		1	2	+7
		1	3	+19
		1	4	+13
		1	5	+16
		1	6	+35
		1	7	+15
		1	8	+3
		1	9	+28

GPSS/PC Report file REPORT.GPS. (V 1.1, # 30840) 07-17-1986 16:05:0 page 6

MATRIX RETRY ROW COLUMN VALUE

	1	10	+4
	1	11	+37
	1	12	+31
	1	13	+1
	1	14	+39
	1	15	+6
	1	16	+38
	1	17	+10
	1	18	+12
	1	19	+29
	1	20	+32

Intermediate values are 0

	2	20 +0

APPENDIX C EXTERNAL MODEL: SIMSCRIPT II.5 PROGRAM

```
)PTIONS  REN=NEW,NOHIST,NOXREF  CACI SIMSCRIPT II.5 IBM S/370 R9.3          PAGE    1
                                                    15-AUG-1986 10:14 (   1)
    1  PREAMBLE
    2   RESOURCES INCLUDE CPU
    3   PROCESSES INCLUDE STATION ,STAT.GEN , SERVER, DISK, WIPE AND CUT.SIMUL
    4   EVERY STATION HAS A NAME, A HOME.SERVER AND OWNS A RESP.BUFFER
    5   EVERY SERVER HAS A STATE, A NUMBER  AND OWNS A REQU.BUFFER AND A
    6   DISK.RESP
    7   EVERY DISK HAS A STATUS, A MY.SERVER AND OWNS A DISK.BUFF
    8   DEFINE RESP.BUFFER, DISK.BUFF AND REQU.BUFFER AS  FIFO SETS
    9   DEFINE NAME,HOME.SERVER,STATE,MY.SERVER AND NUMBER AS INTEGER VARIABLES
   10   TEMPORARY ENTITIES
   11    EVERY REQUEST HAS A GEN.TIME, AN ATTENTION.TIME AND AN ORIGIN, A SIZE,
   12    A SERIAL.NUM, A MODE AND MAY BELONG TO A REQU.BUFFER, THE DISK.BUFF
   13    AND THE DISK.RESP
   14    EVERY RESPONSE HAS A GENER.TIME, AN ATTN.TIME, A COMPLETION.TIME,
   15    A SERIAL.NUM,A SIZE, AND A DESTINATION AND MAY BELONG TO A RESP.BUFFER.
   16    DEFINE SIZE, SERIAL.NUM, ORIGIN AND DESTINATION AS INTEGER VARIABLES
   17    DEFINE GEN.TIME, GENER.TIME, ATTENTION.TIME, COMPLETION.TIME
   18    AND ATTN.TIME AS REAL VARIABLES
   19  DEFINE BUSY TO MEAN 1
   20  DEFINE IDLE TO MEAN 0
   21  DEFINE QREAD TO MEAN 1
   22  DEFINE QWRITE TO MEAN 0
   23  DEFINE QREAD.CLEAN TO MEAN 2
   24  DEFINE QWRITE.CLEAN TO MEAN 3
   25  DEFINE SECONDS TO MEAN DAYS
   26  DEFINE MILLISECONDS TO MEAN HOURS
   27  DEFINE MICROSECONDS TO MEAN MINUTES
   28  DEFINE HIT.RATIO AS AN INTEGER VARIABLE  ** FROM 0 TO 100 PERCENT
```

```
29    DEFINE BUFR.SIZE, NUM.OF.STATIONS AND NUM.RESP AS INTEGER VARIABLES
30    DEFINE IAT.TIME, TURNAROUND.TIME AND EXEC.TIME AS REAL VARIABLES
31    DEFINE REQ.NUM, REQ.LENGTH, DISKQU AND RUN.NUM AS INTEGER VARIABLES
32    TALLY AVG.TURN.TIME AS THE AVERAGE OF TURNAROUND.TIME
33    ACCUMULATE SER.UTIL AS THE AVERAGE OF STATE
34    ACCUMULATE BUFF AS THE AVERAGE OF BUFR.SIZE
35    ACCUMULATE QDISK AS THE AVERAGE OF DISKQU
36  END

OPTIONS  REN=NEW,NOHIST,NOXREF  CACI SIMSCRIPT II.5 IBM S/370 R9.3          PAGE    2
                                                    15-AUG-1986 10:14 (   1)


 1  MAIN
 2    CREATE EVERY CPU(1)
 3    LET U.CPU(1) = 1
 4    FOR RUN.NUM = 1 TO 15 DO
 5    READ HIT.RATIO, NUM.OF.STATIONS AND REQ.LENGTH
 6    LET HOURS.V = 1000
 7    LET MINUTES.V = 1000
 8    LET REQ.NUM = 0
 9    ACTIVATE A STAT.GEN NOW
10    START SIMULATION
11    LET TIME.V = 0.0
12    RESET TOTALS OF TURNAROUND.TIME, BUFR.SIZE, DISKQU AND STATE
13    LET REQ.NUM = 0
14    LET NUM.RESP = 0
15    LOOP
16  END  '' OF MAIN

OPTIONS  REN=NEW,NOHIST,NOXREF  CACI SIMSCRIPT II.5 IBM S/370 R9.3          PAGE    3
                                                    15-AUG-1986 10:14 (   1)
 1  PROCESS STAT.GEN
 2   DEFINE THIS.SERVER, THIS.STATION AND KK AS INTEGER VARIABLES
 3   ACTIVATE A SERVER CALLED THIS.SERVER NOW
 4   LET NUMBER(THIS.SERVER) = THIS.SERVER
 5   FOR KK = 1 TO NUM.OF.STATIONS DO
 6    ACTIVATE A STATION CALLED THIS.STATION NOW
 7    LET HOME.SERVER(THIS.STATION) = THIS.SERVER
 8    LET NAME(THIS.STATION) = THIS.STATION
 9   LOOP
10  END  ''STAT.GEN

PTIONS  REN=NEW,NOHIST,NOXREF  CACI SIMSCRIPT II.5 IBM S/370 R9.3          PAGE    4
                                                    15-AUG-1986 10:14 (   1)
 1  PROCESS CUT.SIMUL
 2  PRINT 13 LINES WITH NUM.OF.STATIONS, REQ.LENGTH,
 3       REQ.NUM/TIME.V,AVG.TURN.TIME*1000 ,SER.UTIL*100,
```

```
      "       BUFF, QDISK AND DISKQU THUS

A SIMULATION OF A SINGLE SERVER SYSTEM
======================================

NUMBER OF STATIONS WAS **
REQUEST LENGTH FOR THIS RUN WAS **** BYTES
THE REQUEST GENERATION RATE WAS    ****.** REQUEST PER SECOND
THE AVERAGE RQUQEST TURNAROUND TIME WAS ****.** MILLISECONDS
THE AVERAGE SERVER UTILIZATION WAS ***.**  PERCENT
THE AVERAGE SIZE OF THE SERVER QUEUE IS **.** REQUESTS
THE SIZE OF THE DISK QUEUE IS              **.** RECORDS
    5   END

OPTIONS  REN=NEW,NOHIST,NOXREF  CACI SIMSCRIPT II.5 IBM S/370 R9.3          PAGE     5
                                                    15-AUG-1986 10:14 (  1)

    1   PROCESS STATION
    2     DEFINE MY.RESPONSE, KK, THIS.REQUEST AS INTEGER VARIABLES
    3     WAIT UNIFORM.F(0.1,0.25,3) SECONDS  ..TO PREVENT CLOGGING THE SERVER
    4     WHILE NUM.RESP < 2000
    5     DO
    6       WORK UNIFORM.F(100.0,150.0,9) MILLISECONDS .. BETWEEN BURSTS
    7       FOR KK = 1 TO 100 DO          BURST LENGTH IS 100 REQUESTS
    8       CREATE A REQUEST CALLED THIS.REQUEST          TIME BETWEEN REQUESTS
    9       ADD 1 TO REQ.NUM
   10       LET MODE(THIS.REQUEST) = QREAD          .. TESTING READ  REQUESTS
   11       LET SIZE(THIS.REQUEST) = REQ.LENGTH
   12       LET GEN.TIME(THIS.REQUEST) = TIME.V
   13       LET ORIGIN(THIS.REQUEST) = NAME(STATION)
   14       LET SERIAL.NUM(THIS.REQUEST) = REQ.NUM
   15       WORK UNIFORM.F(4.0,5.6,2) MILLISECONDS  .. TO PREPARE THE REQUEST
   16       WORK SIZE(THIS.REQUEST)*0.0133 + UNIFORM.F(1.8,3.6,8) MILLISECONDS
   17       FILE THIS THIS.REQUEST IN REQU.BUFFER(HOME.SERVER)
   18       LET BUFR.SIZE = N.REQU.BUFFER(HOME.SERVER)
   19       IF STATE(HOME.SERVER) = IDLE
   20       REACTIVATE THE SERVER CALLED HOME.SERVER NOW
   21       ALWAYS
   22       SUSPEND
   23       REMOVE THE FIRST MY.RESPONSE FROM THE RESP.BUFFER  .. WAIT FOR RESPONSE
   24       IF DESTINATION(MY.RESPONSE) IS NOT EQUAL TO NAME(STATION)
   25         PRINT 1 LINE THUS
ERROR - RESPONSE HAS BEEN SENT TO THE WRONG STATION !!!
   26       ALWAYS
   27       LET TURNAROUND.TIME = TIME.V - GENER.TIME(MY.RESPONSE)
   28       LET EXEC.TIME = TIME.V - ATTN.TIME(MY.RESPONSE)
   29       DESTROY THE RESPONSE CALLED MY.RESPONSE
   30       LOOP
   31     .. WAIT EXPONENTIAL.F(IAT.TIME,1) MILLISECONDS  MAKE IT FREE-RUNNING !
   32     LOOP
                                                                          ..
```

```
   34  END

OPTIONS  REN=NEW,NOHIST,NOXREF CACI SIMSCRIPT II.5 IBM S/370 R9.3          PAGE     6
                                                          15-AUG-1986 10:14 (   1)

     1  PROCESS SERVER
     2    DEFINE NEW.REQUEST, MY.DISK AND NEW.RESPONSE AS INTEGER VARIABLES
     3    DEFINE PRTIME AND COMTIME AS  REAL VARIABLES
     4    ACTIVATE A DISK CALLED MY.DISK NOW
     5    LET MY.SERVER(MY.DISK) = NUMBER(SERVER) **TIE SREVER AND DISK TOGETHER
     6   'SLEEP' LET STATE(SERVER) = IDLE
     7    WAIT 0.1 MILLISECONDS  ** TO LET THE DISK INITIALIZE
     8    SUSPEND
     9    LET STATE(SERVER) = BUSY
    10    WHILE REQU.BUFFER(SERVER) IS NOT EMPTY,
    11    DO
    12     REMOVE THE FIRST NEW.REQUEST FROM REQU.BUFFER(SERVER)
    13     IF MODE(NEW.REQUEST) = QWRITE
    14       IF HIT.RATIO >= RANDI.F(1,100,5) **IS RECORD IN CACHE ?
    15        WORK 1.0 MILLISECONDS ** TO INITIALIZE THE DISK
    16        FILE THIS NEW.REQUEST IN THE DISK.BUFF(MY.DISK)
    17        ADD 1 TO DISKQU
    18        IF STATUS(MY.DISK) = IDLE
    19          REACTIVATE THE DISK CALLED MY.DISK NOW
    20          WAIT 0.1 MILLISECONDS  ** TO LET THE DISK START
    21        ALWAYS
    22        CALL EXEC GIVING NEW.REQUEST
    23       ELSE  **RECORD NOT IN CACHE
    24         IF DISKQU > RANDI.F(0,500,5) **PICKED A DIRTY BLOCK
    25           LET MODE(NEW.REQUEST) = QWRITE.CLEAN
    26           SUBTRACT 1 FROM DISKQU
    27         ELSE
    28           CALL EXEC GIVING NEW.REQUEST   ** SEND RESPONSE IMMIDIATELY
    29         ALWAYS
    30         FILE THIS NEW.REQUEST IN THE DISK.BUFF(MY.DISK)
    31         ADD 1 TO DISKQU
    32         IF STATUS(MY.DISK) = IDLE
    33           REACTIVATE THE DISK CALLED MY.DISK NOW
    34           WAIT 0.1 MILLISECONDS  ** TO LET THE DISK START
    35         ALWAYS
    36       ALWAYS
    37     ELSE  ** MODE OF THIS NEW.REQUEST IS READ
    38       IF HIT.RATIO >= RANDI.F(1,100,6) **RECORD IS IN CACHE
    39        CALL EXEC GIVING NEW.REQUEST
    40       ELSE  **RECORD NOT IN CACHE
    41        IF DISKQU > RANDI.F(0,500,5) **PICKED A DIRTY BLOCK FOR REPLACEMENT
    42          LET MODE(NEW.REQUEST) = QREAD.CLEAN
    43        ALWAYS
    44        FILE THIS NEW.REQUEST IN THE DISK.BUFF(MY.DISK)
    45        IF STATUS(MY.DISK) = IDLE
    46          REACTIVATE THE DISK CALLED MY.DISK NOW
    47        ALWAYS
    48        WAIT 0.1 MILLISECONDS  ** TO LET THE DISK START AND SOME EXEC.
    49       ALWAYS
```

```
   50    ALWAYS
   51   LOOP    ** WHILE REQUEST ARE WAITING IN THE REQU.BUFFER
   52    WHILE DISK.RESP IS NOT EMPTY,
   53    DO        ** TAKE CARE OF REQUESTS COMING BACK FROM DISK
   54    REMOVE THE FIRST NEW.REQUEST FROM THE DISK.RESP(SERVER)
   55    IF MODE(NEW.REQUEST) = QREAD

PROCESS SERVER
UPTIONS  REN=NEW,NOHIST,NOXREF   CACI SIMSCRIPT II.5 IBM S/370 R9.3          PAGE     7
                                                          15-AUG-1986 10:14 (  1)
   56     CALL EXEC GIVING NEW.REQUEST   ** PROCESS REQUEST AND SEND MESSAGE
   57    ELSE IF MODE(NEW.REQUEST) = QREAD.CLEAN
   58     LET MODE(NEW.REQUEST) = QREAD
   59     FILE THIS NEW.REQUEST IN THE DISK.BUFF(MY.DISK)
   60     IF STATUS(MY.DISK) = IDLE
   61       REACTIVATE THE DISK CALLED MY.DISK NOW
   62     ALWAYS
   63    ELSE IF MODE(NEW.REQUEST) = QWRITE.CLEAN
   64     LET MODE(NEW.REQUEST) = QWRITE
   65     FILE THIS NEW.REQUEST IN THE DISK.BUFF(MY.DISK)
   66     ADD 1 TO DISKQU          ** ONE MORE BLOCK IS DIRTY
   67     IF   STATUS(MY.DISK) = IDLE
   68       REACTIVATE THE DISK CALLED MY.DISK NOW
   69     ALWAYS
   70     CALL EXEC GIVING NEW.REQUEST
   71     ALWAYS
   72    ALWAYS
   73   ALWAYS
   74   LOOP
   75
   76    GO TO 'SLEEP'
   77   END

PTIONS   REN=NEW,NOHIST,NOXREF   CACI SIMSCRIPT II.5 IBM S/370 R9.3          PAGE     8
                                                          15-AUG-1986 10:14 (  1)
    1   SUBROUTINE EXEC GIVEN THE.REQUEST
    2   DEFINE THE.REQUEST AS AN INTEGER VARIABLE
    3   DEFINE NEW.RESPONSE AS AN INTEGER VARIABLE
    4   DEFINE PRTIME AND COMTIME AS REAL VARIABLES
    5    LET PRTIME = UNIFORM.F(1.5,1.8,4) + (SIZE(THE.REQUEST))*0.0020
    6    LET COMTIME = (SIZE(THE.REQUEST))*0.0133 + UNIFORM.F(2.0,3.0,5)
    7    LET BUFR.SIZE = N.REQU.BUFFER(SERVER)
    8    CREATE A RESPONSE CALLED NEW.RESPONSE
    9    LET GENER.TIME(NEW.RESPONSE) = GEN.TIME(THE.REQUEST)
   10    LET ATTN.TIME(NEW.RESPONSE) = ATTENTION.TIME(THE.REQUEST)
   11    LET COMPLETION.TIME(NEW.RESPONSE) = TIME.V + PRTIME
   12    LET DESTINATION(NEW.RESPONSE) = ORIGIN(THE.REQUEST)
   13    LET SERIAL.NUM(NEW.RESPONSE) = SERIAL.NUM(THE.REQUEST)
   14    FILE THIS NEW.RESPONSE IN RESP.BUFFER(DESTINATION(NEW.RESPONSE))
   15    IF MODE(THE.REQUEST) IS NOT EQUAL TO QWRITE
   16      DESTROY THE REQUEST CALLED THE.REQUEST
   17    ALWAYS
   18    WORK PRTIME MILLISECONDS
```

```
19    REACTIVATE THE STATION CALLED DESTINATION(NEW.RESPONSE) IN COMTIME
20    MILLISECONDS      ** COMMUNICATION TIME
21    ADD 1 TO NUM.RESP
22    IF NUM.RESP = 2000
23      ACTIVATE A CUT.SIMUL NOW
24    ALWAYS
25  RETURN
26  END    ** OF SUBROUTINE EXEC

OPTIONS  REN=NEW,NOHIST,NOXREF  CACI SIMSCRIPT II.5 IBM S/370 R9.3          PAGE     9
                                                    15-AUG-1986 10:14 (   1)

    1  PROCESS DISK
    2    ** THIS PROCESS SIMULATES THE OPERATION OF THE DISK CONTROLLER
    3    ** WHICH IS AN EXTERNAL DEVICE AS FAR AS THE SERVER'S O.S IS CONCERNED
    4  DEFINE DE.REQUEST AS AN INTEGER VARIABLE
    5  'SUSP' LET STATUS(DISK) = IDLE
    6     SUSPEND
    7     LET STATUS(DISK) = BUSY
    8     WHILE DISK.BUFF(DISK) IS NOT EMPTY,
    9     DO
   10     REMOVE THE FIRST DE.REQUEST FROM THE DISK.BUFF(DISK)
   11     IF MODE(DE.REQUEST) = QREAD OR MODE(DE.REQUEST) = QREAD.CLEAN OR
   12     MODE(DE.REQUEST) = QWRITE.CLEAN,
   13      WAIT UNIFORM.F(75.0,95.0,6) MILLISECONDS  ** TO READ FROM DISK
   14      FILE THIS DE.REQUEST IN DISK.RESP(MY.SERVER(DISK))
   15      IF STATE(MY.SERVER(DISK)) = IDLE
   16       REACTIVATE THE SERVER CALLED MY.SERVER(DISK) NOW
   17      ALWAYS
   18     ELSE IF MODE(DE.REQUEST) = QWRITE
   19        SUBTRACT 1 FROM DISKQU
   20        WAIT UNIFORM.F(75.0,95.0,6) MILLISECONDS
   21        DESTROY THE REQUEST CALLED DE.REQUEST
   22        ALWAYS
   23     ALWAYS
   24     LOOP
   25     GO TO 'SUSP'
   26    END

OPTIONS  REN=NEW,NOHIST,NOXREF  CACI SIMSCRIPT II.5 IBM S/370 R9.3          PAGE    10
                                                    15-AUG-1986 10:14 (   1)

    1  PROCESS WIPE
    2   REQUEST 1 CPU(1)
    3   WHILE DISKQU > 0 DO
    4    WORK 7 MILLISECONDS
    5    SUBTRACT 1 FROM DISKQU
    6   LOOP
    7   RELINQUISH 1 CPU(1)
    8  END

OPTIONS  REN=NEW,NOHIST,NOXREF  CACI SIMSCRIPT II.5 IBM S/370 R9.3          PAGE    11
                                                    15-AUG-1986 10:14 (   1)
```

```
 STATUS OF THIS COMPILATION RUN

   178 EXTERNAL NAMES IN THIS APPLICATION;
    26 CONFLICT WITHIN THE 8-CHARACTER LINKAGE-EDITOR CONSTRAINT.
       INDICATED NAME RESOLUTION IS HANDLED THROUGH THE HISTORY FILE.


PTIONS USED - PRINT,NOMAP,NOLET,CALL,RES,NOTERM,SIZE=753664,NAME=**GO
                                                          VS LOADER
  TOTAL LENGTH     1257U
  ENTRY ADDRESS   121688


  SIMULATION OF A SINGLE SERVER SYSTEM
==========================================

UMBER OF STATIONS WAS  1
EQUEST LENGTH FOR THIS RUN WAS  512 BYTES
HE REQUEST GENERATION RATE WAS        34.41 REQUEST PER SECOND
HE AVERAGE RQUQEST TURNAROUND TIME WAS   26.28 MILLISECCNDS
HE AVERAGE SERVER UTILIZATION WAS   9.19  PERCENT
HE AVERAGE SIZE OF THE SERVER QUEUE IS  0.   REQUESTS
E SIZE OF THE DISK QUEUE IS             0.   RECORDS
                  QUEUE AT END.OF.SIM IS   0 RECORDS

 SIMULATION OF A SINGLE SERVER SYSTEM
==========================================

MBER OF STATIONS WAS  2
QUEST LENGTH FOR THIS RUN WAS  512 BYTES
E REQUEST GENERATION RATE WAS        68.40 REQUEST PER SECOND
E AVERAGE RQUQEST TURNAROUND TIME WAS   26.32 MILLISECONDS
E AVERAGE SERVER UTILIZATION WAS  18.27  PERCENT
E AVERAGE SIZE OF THE SERVER QUEUE IS    .00 REQUESTS
E SIZE OF THE DISK QUEUE IS              0.   RECORDS
                  QUEUE AT END.OF.SIM IS   0 RECORDS

 IMULATION OF A SINGLE SERVER SYSTEM
==========================================

BER OF STATIONS WAS  3
UEST LENGTH FOR THIS RUN WAS  512 BYTES
 REQUEST GENERATION RATE WAS       102.33 REQUEST PER SECOND
 AVERAGE RQUQEST TURNAROUND TIME WAS   26.36 MILLISECCNDS
 AVERAGE SERVER UTILIZATION WAS  27.34  PERCENT
 AVERAGE SIZE OF THE SERVER QUEUE IS    .01 REQUESTS
```