Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

RESOURCE AND JOB MANAGEMENT CONCEPTS IN DISTRIBUTED SYSTEMS

Y. Sadgat

1992
CSD-900099

UNIVERSITY OF CALIFORNIA

Los Angeles

Resource and Job Management Concepts
in Distributed Systems

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Computer Science

by

Yigal Sadgat

1990

© Copyright by

Yigal Sadgat

1990

The dissertation of Yigal Sadgat is approved.

Jack Carlyle

Jack Carlyle

L. McNamee

Bennet Lientz

Bruce L. Rothschild

Mario Gerla

Mario Gerla, Committee Chair

University of California, Los Angeles

1990

The dissertation of Yigal Sadgat is approved.

Jack Carlyle

Jack Carlyle

L. McNamee

Bennet Lientz

Bruce L. Rothschild

Mario Gerla

Mario Gerla, Committee Chair

University of California, Los Angeles

1990

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank the members of my committee, Professors Jack Carlyle, L. McNamee, Ben Lientz and Bruce Rothschild. I am especially thankful to my long time advisor, Professor Mario Gerla, whose guidance during this long period of time was invaluable. His creative thinking, patience, enthusiasm and support have been a continual source of motivation and inspiration. I am also thankful to Verra Morgan who has quietly contributed to the success of my research.

Last but not least, my wife has been an important source of encouragement without which it would have been impossible to conduct this work.

VITA

1973	Practical Engineer in Electronics, Syngalowsky Institute, Tel Aviv, Israel.
1973-1977	Electrical Engineer, Air Force Headquarters Computer Lab. Israel.
1977-1980	Electrical Engineer, Digital Equipment Corp.
1981	BS in Electrical Engineering, California State Polytechnic Institute, Pomona, CA.
1981-1984	R&D Engineer, Amdahl Communication Systems Div., Marina Del Rey, CA.
1983	MS in Computer Science, University of California, Los Angeles.
1984-present	R&D Engineering Manager, Xerox Corporation, El Segundo, CA.

ABSTRACT OF THE DISSERTATION

Resource and Job Management Concepts in Distributed Systems

by

Yigal Sadgat

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1990

Professor Mario Gerla, Chair

Many existing office automation systems support a large number of users taking advantage of a multitude of shared services like archiving, printing, mailing, scanning, and computing. Further technology advancements will lead to offering additional-services and features including voice, music, still or moving video, and images addressing multi format issues. Jobs submitted by the users will be more complex and may require the cooperation of several services to complete: printing voice messages is one simple example.

With the ever increasing number of resources and users, resource and job management protocols are needed in order to optimize resource utilization and job assignments in the network while maintaining or increasing transparency, improving response times, and relieving the users from the need of being familiar with the network services.

In concert with the above, this work proposes a flexible approach supported by an entity called the Task Manager (TM). It receives service requests from the users and based on its own network knowledge, partitions the job into tasks and optimally assigns them to servers while minimizing response time and network traffic. The TM may manage any job mixture, a set of one type servers, or a balanced set of different servers. The TM may dynamically change its domain by releasing or acquiring servers. It transparently handles resource failures, removal or addition of network services and depending upon the particular system, the TM may be distributed or centralized. The TM abstract belongs in the Application layer and is capable of accommodating many communication networks.

Simulation was used to measure the performance improvement achieved by implementing a distributed TM in a local area network. Given job size and complexity, the effect of changing arrival rates and workstation count on response times, utilization and queuing times have been studied and possible bottlenecks identified. Response time sensitivity to changing workstation and server processing speeds as well as load balancing algorithms have also been analyzed.

1.0 Introduction

In the past several years local area networks (LANS) [Met76] [Bog80] and distributed systems have gained popularity and drawn the attention of researchers as well as network and distributed system developers. Early networks offered limited functionality and relatively low communication speeds. They allowed for resource sharing, bettered their utilization and improved total system reliability. Today LANS support larger and faster distributed systems offering advanced and more complicated services coupled with improved diversification at a much lower cost. The number of users supported has risen and additional environments have been addressed.

The concept of networking and distributed systems has proliferated to many areas and is not limited to the scientific or academic environments [Pop81] [Wal83] [Nel84] [Lis83] [Lis85]. Many existing office automation LANS support a very large number of users taking advantage of a multitude of shared services like archiving, printing, mailing, scanning, formatting, and computing which are readily available in the office environment. The factories followed with automation standards (MAP etc.) which help integrate a multitude of devices together to support manufacturing tasks.

These devices include control computers, robots, component storage systems, graphic CAD workstations, plotters etc.

Further research in networking and communication coupled with standardization [Zim80] led to wider acceptance of ETHERNET [Met76] based LANS. Distributed system algorithms facilitated further enhancement of applications like process control and distributed data bases (DDBMS) as well as others relying on the cooperation of multiple sites. One of many important requirements from the user's point of view was that distributed and non-distributed systems do not differ from each other with regard to their interfacing with them. Nevertheless the number of different and conflicting user interfaces stands in direct relation to the number of systems offered.

Figure 1.1 depicts a typical LAN and its elements supporting engineering and academic environments. Workstations perform a multitude of tasks including engineering development, office related document creation and editing, electronic mail, accounting, process control and remote network management. The more advanced workstations may support CAD/CAM and other graphically oriented tasks and may contain peripheral units like disks or tape drives. Another service offered is filing and archiving. Each user may have a dedicated file space or access to public files. Its size and location is transparent to the users. Those files not frequently needed may be archived and retrieved when needed. Computation bound tasks may be directed to the departmental processor or alternatively, through the communication

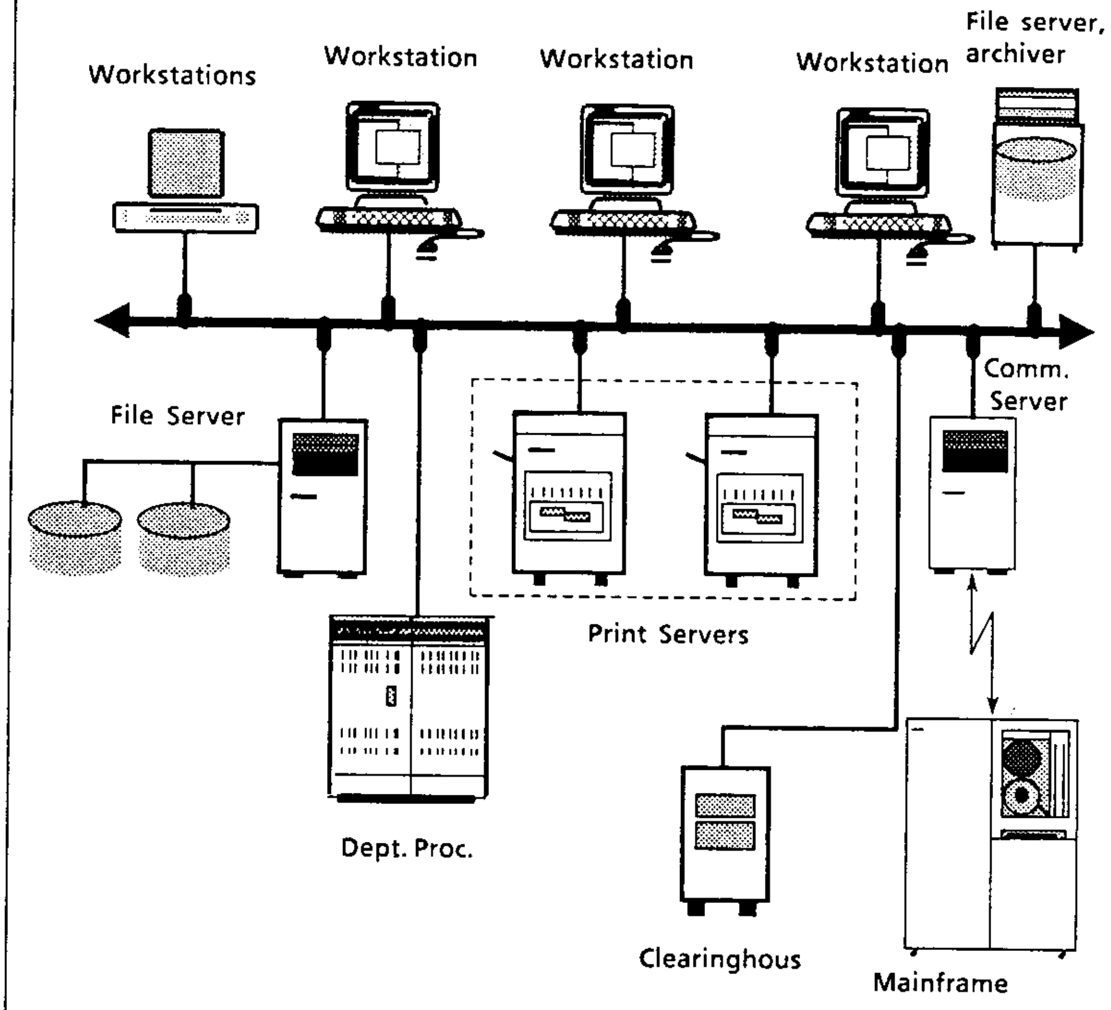


Figure 1.1: Office automation network configuration

server to the remote mainframe shared with other networks as well. The users requiring hard copies may select any of the print servers based on their location and features. In case a printer is unavailable, another one may be selected. A service which is transparent to the casual user is the clearinghouse, also called a name server. Last but not least, all of the above are connected by a LAN at transmission speed of 10MBPS.

Jobs and servers are managed by the users and to some extent by the network administrator. Users decide on job's functional flow thus they need to know the exact steps to be performed as described in this section. They may assign a task to server S_a, upon completion transmit it to server S_b and then to S_c . Servers freely accept requests from users and are not being "managed" by anyone. Authentication is the only function some of them perform. For example, file servers may use an authentication protocol before responding to "read" or "write" requests from users to ascertain that they are who they claim they are and that they are permitted to have the requested access. The management questions become more acute considering new technologies and user sophistication. Users should not be asked to perform system management functions since (a) they are not in a position to do that, (b) it should be transparent to them, and (c) a better candidate may be identified. As the systems grow more complex, more servers become available, and there is a need to address different formats which are not normally addressed by the users and different media which the user may have no control or knowledge of. There is also a need to respond to dynamic changes occurring due to different loads and system faults. These functions become too complex and time consuming.

1.1 The Need for This Work

Some terms used in this work should be defined:

> A *job* is a bounded set of *tasks* $\{t_a, t_b, t_c, \ldots t_n\}$ to be performed on an object.

$$t_a \rightarrow t_b \rightarrow t_c \rightarrow \begin{matrix} t_d \\ t_e \\ t_f \end{matrix} \rightarrow t_k \quad \ldots\ldots\ldots \rightarrow t_n$$

The tasks may be inter-related with some precedence relationships like t_a and t_b while others may take place in parallel: t_d, t_e, t_f. For instance, printing is a job. It may include an object residing somewhere in the distributed system which has to be transmitted to a server capable of performing the needed operation - print. Performing multiple computations is another job.

> *Job Management* defines the process employed in executing a job including partitioning, task assignment, load balancing and exception handling.

Said differently, job management is the step by step process a job undergoes from its reception till successful completion. Job management and *System Management* are related.

> *System Management* is a set of protocols by which the system components are controlled.

Given servers and clients, system management includes task assignment, load balancing, feature matching, utilization optimization and fault tolerance. In order to further clarify the terms above, the following office automation example is provided. A user wishes to print a document he received from a colleague. He has to know its format and encoding and then select a printer by considering its physical location and features. Nowadays, the job management function is done by the user: he submits the job to the selected printer. There is no system management done in this case - there is no knowledge of how loaded the printer is, its status (active or out of service) nor action taken upon successful or unsuccessful job completion.

Servers provide services and no distinction is made between these two terms. Users and clients are also used interchangeably. This work addresses the distributed scientific, engineering and office environments.

All of the distributed systems in the office and engineering environments offer the individual users with the necessary tools to create, manage and exchange textual information without the need to understand how these operations are being executed on their behalf. Once the network logical name of the intended document recipient is known, all the user needs to do is mail the message without, for instance, worrying about routing, fragmentation or transmission errors. Similarly, filing is transparent to them. Physical location and electrical characteristics of the server are issues that the users do not need to consider. Additional tools are also available to them to assist in their day-to-day operation.

The need to share information, perform group activities and use common resources may necessitate knowledge of multi media and format devices if current systems do not improve. Limiting sharing to the immediate environment does not support efficient sharing but relieves the users from considering multi media and format issues. This limitation is unacceptable and ignoring the subject means that users will have to contend with it themselves thus requiring them to get involved with format and media issues which they might have no interest in nor knowledge of. A simplistic approach requires the resource and destination formats and media to be identical. For example, if a text only document is sent, the recipient must be able to: (a) *interpret* the document format, (b) understand its *encoding and character sets*, and (c) have the necessary *character generators/fonts* to either display or print it. Failure to support any of the above will prohibit processing the document. On the other hand, the destination system may use different formats and media leading to the need to translate the format and consider the different media, issues that may not be trivial. Not all formats can be translated. Also, if the original media requires a bit mapped screen at a certain resolution, the destination must support that, otherwise another conversion may be needed. Note that the commercially available systems offer services relying on existing technology in the areas of I/O, processing and communication. Future technologies will offer yet additional capabilities and features at much better price/performance.

With the vast number of applications and system configurations currently available there is a need to provide system and job management functions which will not rely on the user's past knowledge of the network. In XNS [Xer01], for instance, the users have to select a server and monitor its status. Frequently, if the server is busy, they have to wait despite the fact that there might be another server available. In more demanding environments, there is a need to know in advance the server status, capabilities, location, and performance. Furthermore, multi format systems require the users to perform intermediate steps which may be routinely performed by a service on their behalf.

The problem may be compounded if we consider future systems supporting additional features. For instance, mailing text messages to a recipient who has the capability of processing text only is acceptable but messages containing text and graphics will pose a problem. Issues like printing text and graphics can be resolved but printing animation is not as simple and so is the case with printing voice messages.

Current and future systems require additional functionality to reduce the amount of human guidance in:

(a) the process of job submission and

(b) job management following submission.

Very often users rely on their familiarity with the system thus expecting it to perform a routine job the same way over and over again. Possible random system faults have to be addressed by them and corrective measures taken. Adding, removing or performing back-up on file servers interfere with normal operation and should be transparent to them.

Most systems do not provide adequate system and job management functions leading to congestions and possible bottlenecks. In several systems (XNS, DECNET) [Xer01] load balancing and resource selection, in case of more than a single instance of a server, is either non existent or left to the user. The user has to

(a) know about the availability of the particular service and its capabilities, (b) select a server,

(c) inquire the resource's state and status; e.g. is it alive and how long its input queue is,

(d) possibly repeat the sequence for other services in case he is unsatisfied with the outcome of step (c),

(e) decide which one to use, and

(f) submit.

The process may get more complicated considering possible intermediate processing needs like pre or post processing.

Steps (a) to (f) inclusive partially describe the Job Submission Process. Once the job has been submitted to the selected server there might be a need to report its interim and final status. The submitted job may complete with or without errors. Server failures due to a multitude of reasons may lead to unnecessarily repeating intermediate steps that could have been saved otherwise. Intermediate steps like task partitioning, status reporting, and scheduling should be handled by a service based on a pre-defined set of rules thus requiring job management protocols. There is a need to specify it considering: load balancing, feature matching, utilization optimization, transparency, fault tolerance, increasing overall system performance, and cost optimization.

This work will provide resource and job management protocols and concepts for the office, engineering, and scientific environments while improving response times. Improving response time will be verified by simulating implementation in the XNS environment. The simulation will also provide for a performance tradeoff discussion.

1.2 Related Research

The protocols and concepts researched in this work may support the *Digital Library System* (DLS) as proposed in [Kah88]. One of the functional components in the DLS is the *Knowbot* which is similar to the TM as proposed here. The MP in [Gai87] is a distributed process manager for engineering workstations dealing with concurrent processes and process migration with some conceptual similarities with the Task Manager in my work. The RM [Sum87] deals with resource sharing for personal computers in LANs. Andrew is the name given to a distributed personal computing environment [Mor86] in Carnegie-Mellon. This research addresses personal computing and communication systems. As with XNS [Xer01] which has influenced its design, it does not provide for job management.

Digital Library System

The Digital Library System (DLS) [Kah88] is a set of concepts for managing a vast collection of current or transient information that may be available in libraries or archives in the form of video and audio tapes, books, magazines, disks and even newspapers (which are mainly of transient interest). It is viewed as a collection of information and knowledge made up of many DLS's that are geographically distributed and managed by different organizations. Convenient access to local and remote information, regardless of location, is an essential goal of the DLS.

The DLS proposes to deal with the large amount of information already available in computer based form but might not be easily accessed therefore of little use.

A *Personal Library System* (PLS) which is uniquely tailored to the user's individual needs will perform tasks on his behalf. If local requests for information are not satisfied by the PLS, perhaps a more global source of information should be searched thus leading to spawning multiple inquiries dealing with possibly thousands of DLSs. This search is mediated by an active intelligent program called *Knowbot*. One possible Knowbot is a document editing Knowbot. The Knowbots are constructed on behalf of the users to accommodate their requests and operate in their *Knowbot Operating Environment* (KNOE). They communicate with each other by sending messages and may even be used to transport other Knowbots. Depending on whether Knowbots directly serve users, they may be classified as user or system Knowbots. User Knowbots accept instructions from the user and determine how best to meet the stated requirements, perhaps by interacting with other Knowbots. "Certain Knowbots have a permanent status within each user's system and are known as resident Knowbots" [Kah88]. Another kind of knowbots may be spawned dynamically to complete a task and then deleted.

One of the principal concepts in the DLS is the ability to access any document within the entire DLS. Using the PLS, the user can modify a document, combine it with another document, search for phrases in it, mail and then print it. "Two important components of the DLS ... are the Import / Export servers and the Representation Transformation servers. The former components are responsible for accepting new documents into the DLS and for dispatching documents out of the system. The latter components convert documents from one internal representation to another. Depending on the nature of the output required, the obtained results may be passed through a Representation Transformation server for conversion before being delivered" [Kah88]. The results may be destined back to the originating PLS or a target PLS designation outside the particular DLS thus passing through the import export server. The principal components of the DLS include import/export server, registration server, statistics and billing, representation transformation servers and others.

Issues like standards, scanning, digital representation of recorded or synthesized voice, and conversions from old to new formats are being addressed. One of the DLS enablers is standardization to facilitate information sharing between DLSs controlled by different organizations.

The initial application of the DLS will allow users to retrieve documents for which they may be able to supply only an imprecise description. It will also be tailored for the domain of printable documents.

The DLS and the TM are similar in some respects but different in others. The TM allows users to tailor their domain to support individual needs. If a user service request cannot be performed by the TM, due to a transient or permanent loss of capability, the TM will request other TM's help. As with the Knowbots sending messages to each other, the TMs can interrogate and assign tasks to each other. Knowbots and TMs alike accept requests from users and act on their behalf in executing the requests. However, a major difference between them is that TMs do not spawn ancestors and do not generate remote TMs. A TM is statically assigned to a site and can only assign tasks to other TMs with the expectation that the TMs will perform the tasks rather then reassign them to other TMs. The TM local data base is the enabler upon which this expectation is based.

The TM and DLS access the same type of servers. In fact, the initial DLS implementation deals with printable documents. The TM concept also deals with printers and print related activities. The protocols developed in chapter 3 of this work may also help in the DLS research program.

Another related work is the **Distributed Process Manager** (MP) [Gai87] which is a distributed process manager for an Ethernet connection of engineering workstations with the functions of invoking concurrent process sets, downline loading of programs to processing sites and transparent process migration. It manages execution of several processes in parallel on cooperating workstations taking advantage of the multiplicity of workstations. The MP is insensitive to the number of workstations in the network but is limited to the subnet: workstations connected through bridges and gateways are not included in the process management. Workstations in the network compete among themselves to respond to requests for processing since the MP does not assign jobs to them but rather waits for them to respond to a multicast message thus implementing receiver initiated load sharing. To the entities using its services, the MP appears as a single process despite having being replicated in every workstation as a shadow MP. It manages process migration and load sharing transparent to the users.

The MP and TM share several properties, one of which is providing failure and location transparency. As with TM failures (without history files) that may lead to the loss of jobs already executing in the system servers, the MP may lose context when a parent process station fails. As with the TM fault tolerant protocols, if the principal MP fails, one of the shadows takes over and deals with orphan processes. MPs exchange information in form of status, commands and argument lists as well as data.The MP and the TM are both application level oriented.

The **RM** system, described in [Sum87] [Sum89], is an experimental prototype that supports the use of distributed services offered by personal computers connected to a local area network. Using a service request model, RM allows any PC on the LAN to offer and use services. Once RM is initiated, the users are provided with local and remote actions: the Disk Operating System (DOS) commands are executed locally and the RM actions executed remotely on other PCs offering remote services. To invoke a service offered by any machine, users issue a SendRequest call. If the request is successful, a requestID is returned from the remote server. This is done by the server sending a Receive Request message with the ID. Although the user (or a program on his behalf) may request service, the user himself is responsible for selecting the server, receiving results and if needed, resubmitting the task to another server for further processing. No load balancing is performed and transparency (i.e. users should not need to submit their jobs to more than one server regardless of job complexity as described in the next two chapters) is not supported.

The **Cipon** model [Wag86], like the RM, concentrated on how a set of local systems can be transformed into a distributed system. A set of one type of local resources is combined to form a distributed resource of the same type. These clusters of resources are managed by a resource manager. One of the requirements was that the interface to local resources should not differ from remote resources therefore maintaining transparency. A communication model is developed to provide for weakly coupled processes the capability of master/slave status. It has been implemented for personal workstations and uses the ISO transport layer with remote procedure calls. The Cipon system supports functions needed in the office environment: printing, filing, mail, time-of-day and others. It is used to build a transparent, distributed and network-wide file system from local files on each station. Like the RM, it doesn't provide system and job management.

Andrew is a distributed personal computing environment [Mor86]. It is based on four key components: high bandwidth network, personal computers, raster graphics and time sharing file systems. Andrew, which is strongly influenced by Xerox Network Systems [Xer01], uses workstations called Virtue, Ethernet and fiber optic LANs supporting a file system called VICE. The basic architecture of VICE has only simplified support for services such as mail and printing. Printing is merely a file transfer to the print server. Mailing consists of storing a file in the mailbox subdirectory. If users require multiple step processing in order to complete a job, they need to submit multiple service requests until all needed services are completed and the job is done.

Telesophy. Telesophy's [Sch85] purpose is somewhat similar to the DLS: creating a system providing transparent knowledge manipulation for different types of data stored in different physical locations. A universal library may be connected by a network called WorldNet: all of the world's information may be available through this network regardless of media type and location. Telesophy or "wisdom at a distance" should be available to anyone without the need to possess the knowledge of where to search for the information. The system components included in this work are multi format databases and the worldwide network. Navigation through the databases ("information space") requires a continuous interactive series of fetches from information distributed throughout the network. Once the information is retrieved, it is manipulated by the user and stored as *Information Units* (IU). Thus in a Telesophy system, the user sees a single large collection of nodes: the IUs. In the current prototype, since the WorldNet is unavailable, a set of 20 databases are incorporated and all database search queries repetitively interrogate them. File type, format and location transparency is required in order to support "fast transparent worldwide access to information" [Sch87a].

Telesophy and the TM share some concepts: transparency, and unified information (or service) space. However, Telesophy is brought here as an example of how the TM concept may be incorporated in it: users may assign their tasks to the TM rather than perform them themselves.

2.0 Functional Requirements

The previous section discussed the need for system and job management in distributed systems. This section discusses the functional requirements from the user's point of viewed as well as added global requirements in order to tailor the research model to suit these needs. The users who interface with the system may view it as a "black box". Nevertheless, there are very demanding requirements which are not easy to quantify (how can transparency be quantified?). The requirements are categorized based on their effect on the user or the "system". Some requirements fit both.

2.1 The Environment and Its Characteristics

The environment is qualitatively discussed in order to provide the context for the functional requirements. Three entities make the environment:

(a) the **users,**

(b) the **services** provided, and

(c) the **interconnection** network.

2.1.1 The Users

Several user types may be identified. The **layman** is characterized by repetitive use of a small set of services. His system/global demands are minimal and so is his inner-working knowledge of it. Although the number of services needed by one individual is small, not all laymen use the same set of services. Therefore there is a need to: (a) provide him with enough on-line information to facilitate his use of the system, and (b) protect the users and the system and recover from possible mistakes. Supporting laymen does not reduce the complexity of the issues. User requirements may change in time.

In contrast with the layman is the **expert** user. System developers, engineers, scientists and researchers belong in this category. They may be intimately familiar with the system operation and demand a multitude of services to be provided in addition to low response time. They also err thus needing the tools to recover without affecting other users. The experts also tend to belong in the group of frequent users. Regardless of how frequent users access the services, their level of expertise and service needs may change in time.

Although the user interface is outside the scope of this work, the equipment used has to be considered. Input devices include the following traditional as well as advanced means: keyboards are used for text; pointing devices like mouse etc. are becoming available especially in graphic oriented applications; scanners at different speed and resolution provide bit mapped images; magnetic and optical disks and tapes have been used for mass storage and backup, voice input as well as video (digitized and possibly compressed) is being offered. Output devices include the popular CRT for text and graphic images either in monochrome or color; various printers and plotters as well as laser printers supporting page description languages; various disks and tapes for filing and archiving; voice and video signals in digital form may be recorded and/or transmitted over the communication lines.

2.1.2 The Services

The intent of this section is to point to the various categories of services, not to list them or discuss their technology. Each server is managed by control messages and receives and/or generates data. Servers like printers receive data only thus they belong in the *input-only* category [Wag86]. Scanners convert printed material into electronic images therefore they belong in the *output-only* category. Computing services, for example, receive and produce data, thus belonging in both categories. They all will be included in the protocol sets proposed in this work regardless of their function or performance. These services include filing, name servers, printing, mailing, format converters, voice, and video servers. Computing services may be available to all users but disk filing space may be restricted. These servers are not controlled: tasks assigned to them are random based on a first come first serve basis and in fact their logical location in the network is of no interest to the users. Some servers are transparent to the casual users: name servers, for example. It is anticipated that the number and variety of services will always grow.

2.1.3 The Interconnection

The third element in the environment is the interconnection mechanism which may be a LAN, WAN, FOLAN etc. The network is being viewed generically and characterized by several parameters including transmission speeds ranging from 2M bps in twisted pairs to over 100M bps in FOLANS. Different standards (e.g. 802.2, 802.3, ISO, ODA, XNS[Xer01]) coupled with transmission protocols (e.g. TCP/IP, XNS) may be used. The communication system implementing different standards and protocols provides for the necessary point to point connectivity. A basic assumption is that the underlying network provides for connectivity while maintaining error free transmission between the sender and recipient. The network should operate reliably but there is no requirement that (internally to the network) errors should never occur. Messages are transmitted in sequence. Routing, duplicate suppression and other related topics are transparent to us.

2.2 User Needs

This section describes the needs from the point of view of the user.

2.2.1 Transparency. It is believed that the most important need is that the users will not be expected to know the inner-working of the system nor the various steps needed to accomplish their tasks [Wal83] [Sch85] [Kah88] [Orr88]. Hence there is a strong requirement to provide for **type** and **format** transparency [Sch85]. Users need to manage all of their jobs the same way regardless of their type: text, graphics, voice, video, etc. [Ter87]. The second requirement is to provide for format transparency meaning that the different jobs should not be handled based on their format. Both type and format transparency relieve the users from the need to (a) **know** and **track** these parameters, (b) know **how** to process them and **what** can be employed to do it, and (c) know the system **dynamic status** to optimize these operations. Type and format transparency is also needed to support the various media comprising the network. It is anticipated that even with the same media type, file formats will be different due to the integration of different standards in the operating environment. The third requirement is **location** transparency [Sch85] which may be further broken into two parts: (a) name transparency and (b) response time. The first is not addressed in this work while the second is covered in 2.2.3

2.2.2 Job Partitioning and Task Assignment. Job partitioning and task assignments have been discussed in several papers [Chu80]. Users should not be asked to perform either one. Not all servers are created equal. Features and performance offered by them are not identical. In addition, not all tasks submitted require the same processing. Users should not be forced to decide on task assignments since they do not have the knowledge nor the tools to optimize the selection. For example not all print servers support the same page description language. In case a task needs stapling, it should be assigned to a printer capable of stapling.

2.2.3 Response Time. Users, naturally, would not like to wait for their jobs regardless of how complex they are. Transparency, as described in 2.2.1, offloads tasks from the users but may increase server load thus creating a potential problem: users may need to wait for the system to complete jobs that used to take their own time. In addition, the response time may not be uniform due to the physical partitioning of the network. Assume a file server S_1 and a user U_1 are connected to the same network segment as depicted in figure 2.1. This network is made of several Ethernets and inter network routers. User U_{n+2} who is connected to a different segment may need to access S_1 through the network router. User U_1 does not need to pass through the network router. Since routers are slower than the Ethernet, the response times will be different thus affecting location transparency.

2.2.4 Availability. Availability includes **fault tolerance** and **transmission error** recovery. Since in this work the interest is in the higher levels of the

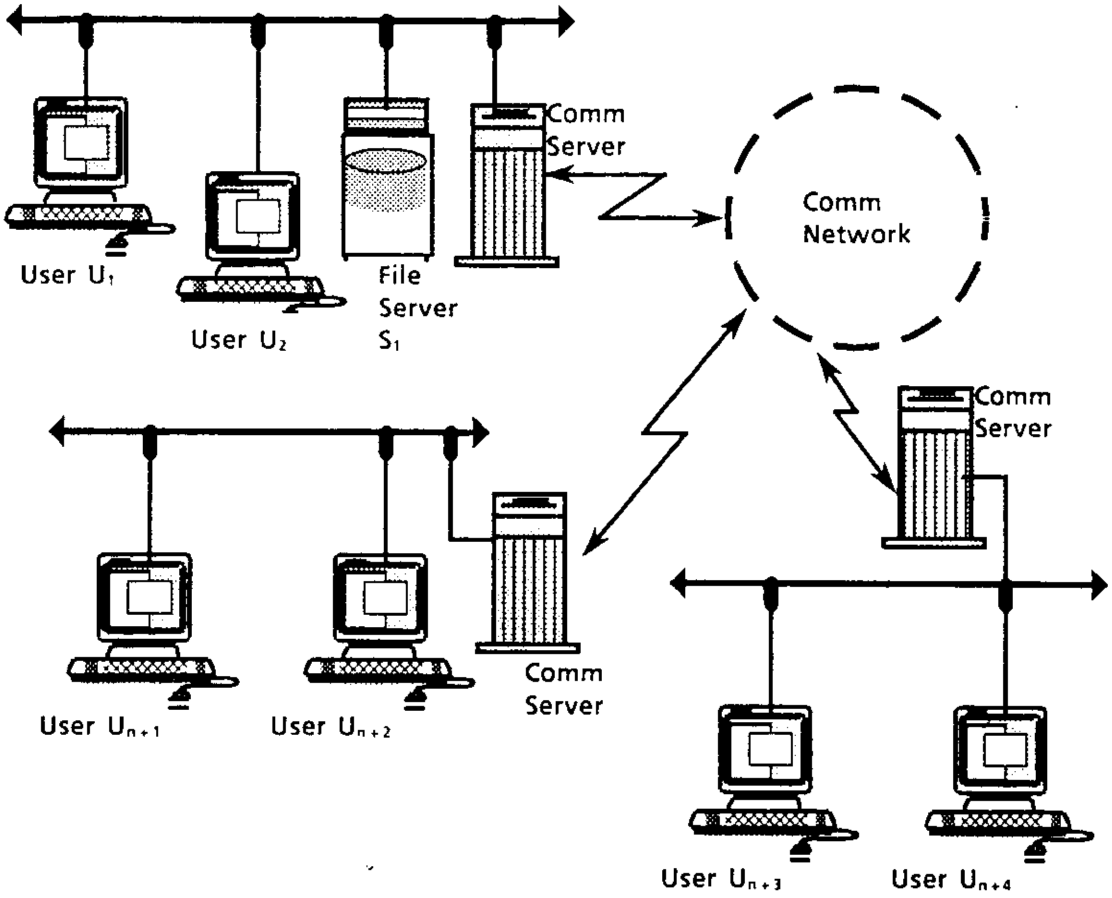


Fig. 2.1: Network segments

communication protocols (Application) it is required that the communication protocols provide error free transmission. It is not required that the servers will never fail. The protocols should provide recovery from server failures. Furthermore, it is required that failures should not affect the users, i.e. users should not have to re-submit their tasks due to a system failure. "Done" notifications may be sent back to the users once their jobs are complete. The addition of the protocols proposed in this work should increase the overall system availability by, for instance, directing jobs to functioning servers and even redirection upon unsuccessful termination. The system should be able to automatically detect and recover from failures without prolonged effect on performance. This may require servers to communicate and exchange "are you alive" packets [An85]. Server failures should be detected and corrective action must be taken to recover from them. Users <u>may</u> be required to re-submit their jobs. If a task cannot be completed because of a server failure, other servers should be selected. Another alternative is to partition the job differently and select a different assignment.

2.2.5 Statistics collection. There might be a need to collect statistics regarding material and time usage per user and server as well as statistics like utilization rates. The information is useful for better network planning and billing.

2.2.7 Job Manipulation Following Submission. Users should be able to delete their jobs or inquire about their status.

2.3 System Needs

The user requirements strongly reflect on the system. There are additional requirements that may be transparent to the users and are covered in this section. These requirements do not restrict the users.

2.3.1 Load Balancing. Tasks should be allocated to servers in such a way that they will all be busy: there will not be a situation where a server is idle when it can perform the same job directed to another busy server [Arv89] [dSe84]. The intent is to reduce average system response times.

2.3.2 Server Utilization Optimization, Cost. The servers should be supported in identifying potential job related problems, and in resolving them **prior to job execution**. Servers may abort jobs containing "Include" files which are not local since they may not be able to "remote fetch" them. In some cases, jobs may need to be aborted due to lack of data or attributes. Servers should always be busy and prevented from waiting for dependencies. In case server S_1 possesses capabilities {a,b,c} and server S_2 possesses {a} only, a task requiring {a} may be performed by S_1 and S_2. In a simple case in which both servers are idle, it is preferred to assign the task to S_2 in anticipation of incoming tasks that may require {b, c}. In commercial systems, cost should also be considered. For instance, if storage costs are different between two sets of file servers, everything else being equal, the least expensive server may be used.

2.3.3 Feature Matching. The jobs submitted to the system may have different requirements. Print tasks may need different types of finishing or be represented by different presentation versions. The system should direct the tasks to the servers which are capable of executing them. Furthermore, if there is no such server available, the task should be partitioned into subtasks that can be executed by the servers or alternatively "export" it to other servers for help.

3.0 The Solution Approach

The previous sections described the environment, requirements and problem definition. This section details the research solution. Sections 3.1 and 3.2 describe object attributes, job submission and management leading to the proposed solution detailed in the rest of this chapter.

3.1 Objects and Object Attributes

Throughout the description a distinction is made between the "attributes" and the "objects" [Wag86] therefore these terms should be described.

> *The "object" is the detailed task description:* a program to be executed, the data file that describes the "marks on paper" in a page description language or any other form if the task is print related, encoded video signals describing the picture if dealing with still or moving pictures, electronic voice messages, etc.

The object may take any format. It can be created by programmers in any programming language or be generated by other objects as well as system servers like scan-cut-paste. Objects are static thus do not depend on the current execution request, i.e. they represent permanent information.

The "attributes" are the task requirements: number of copies to make, disk space needed, services needed to successfully accomplish the task, encoding standard used, language description ('C', fortran, etc.), include files, type, length, external routines, etc.

There are two attribute types: *static* and *dynamic*. The static attributes are those which do not change in time nor depend on the execution instance. The encoding standard of an object is static, its interface requirements and file type are static. If there is an attribute denoting the programming language used in this source code, it would also be static. Dynamic attributes directly relate to the current execution request. In case of printing: number of copies to make, material needed, execution start time, which entity should receive its output etc. In summary: some of the attributes are object related and are static while some are current request related and are dynamic. Although objects and attributes are distinct entities, they may be manipulated simultaneously. All objects should have associated attributes but in case they are missing, some attributes may be constructed with information obtained from the object itself. For instance, assume that "include" file names have to be specified in the attributes, but the object containing these include names has been submitted without attributes. In this case, the needed attributes may be constructed by scanning the object. It is permitted to have objects solely made of file names.

3.2 Job Submission and Management

In distributed (and centralized) systems several routine steps are being performed:

(a) job submission by the users and

(b) job manipulation within the system once submitted.

A distinction is made between the job submission and its management once submitted.

3.2.1 Job Submission

Definition: *Job Submission is the set of protocols governing data and control transfers between the user and a predefined object in the system.*

It addresses the submission process and may be composed of the following steps:

(a) The user identifies the processing step to be taken knowing the capability and availability of several servers.

The user decides on the next step to be performed: print, mail, archive, file etc. There is a need to know which servers are available and furthermore, their capabilities have to be known: disk space, the ability to handle multi format files, vector processing capability, stapling capability, etc. Storing classified information in a private file server, for example, may not need encription, but storing it in a public server may need the extra step to prevent unauthorized reading. Note that not all 'available' servers are available to everyone on the network at all times due to failures etc.

(b) The user pre-selects a server.

Based on the above, only one server is picked to perform the task.

(c) The user inquires about the server's status.

The server may be out of service and/or its input queue - very long. Additional information received by the user may include free disk space, paper type and amount (for printers), statistics, capabilities, etc. In case the user is unfamiliar with the server's capabilities, this step will provide the needed information.

(d) The user may possibly repeat the above steps.

Unsatisfactory results may necessitate selecting a replacement server.

(e) The user selects a server.

Based on the previous steps, he has the necessary and sufficient information to make the 'right' selection.

(f) The user submits the job.

He may possibly follow up as well.

The environment in which the above steps may be taken is depicted in figure 3.1 which includes users - also referred to as clients, file servers, printers, compute servers - also referred to as departmental processors, conversion servers, and the communication network. This does not preclude other servers like electronic and voice mail and scanning.

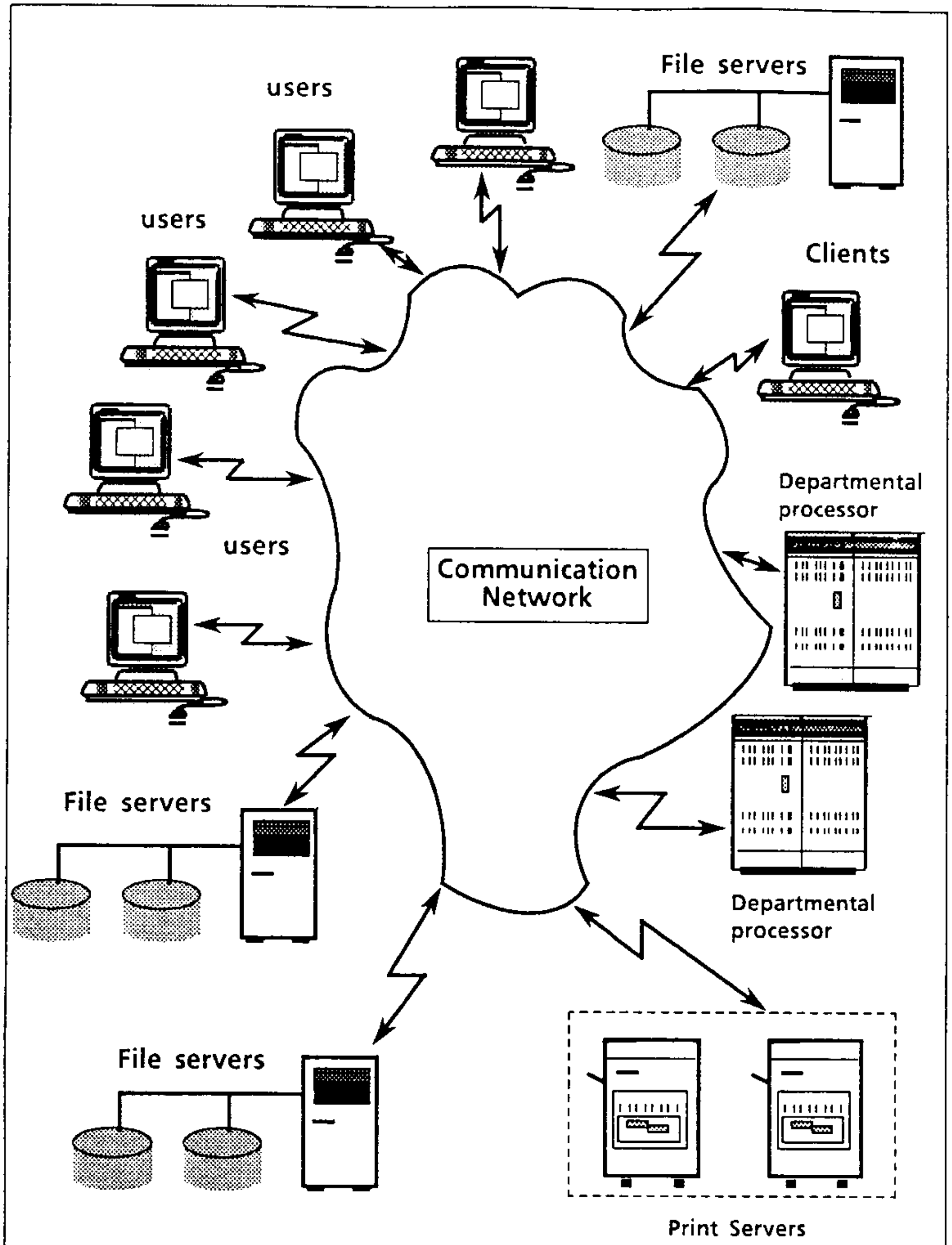


Fig. 3.1: Users, servers and the communication network

The six step process described above should also address the following issues:

(a) Load balancing: all the tasks in the network should be 'equally' divided between the servers to reduce system response times, a task not easily performed by end users.

(b) Server utilization optimization and feature matching as described in 2.3.2.

(c) Multi media and format. These considerations are critical. Mailing a document adhering to a certain format not understood by the recipient is useless. Requiring certain media not available to the recipient is yet another problem.

Note that these issues should be transparent to the users and not be performed by them.

3.2.2 Job Management

Definition: *Job Management is the set of protocols governing data and control transfers between all network servers.*

It is assumed that jobs submitted to the system may be processed locally or submitted to a server after which they may need additional processing thus be re-submitted to other servers until they leave the system or terminate. Partitioning the job into tasks, task assignment, and follow up as well as other possible functions covered in section 3.3 compose job management. The set of job management protocols deals with internal data and control transfers between network servers while the submission protocol deals with the way users interact with the network. The previous section described the steps taken by the user to submit his tasks. The management protocols relieve the users from performing those tasks and yet provide additional services.

3.3 The Task Manager

The requirements listed in chapter 2 (increased functionality, transparency, job and resource management, etc.) can be met by assigning the tasks described in section 3.2 to a proxy: the **Task Manager (TM)**. The goal of relieving the user from the responsibility of performing these tasks may be achieved immediately. The TM receives the object, its attributes and general information unique for each user, as described later in this section, in order to be independent. Being a dedicated function it may perform the tasks on behalf of the users, dynamically collect network information to better facilitate its job management, address additional functions which cannot be performed by the users and even increase availability, as discussed later. It will be logically placed between the users and the needed services thus all service requests will be directed to it as depicted in figure 3.2

3.3.1 The TM Concept

The TM may manage a unique set of tasks: print only, file only, mail only, etc. The tasks assigned to it will only require the processing of the servers available to it; exception handling is discussed later. Example: print tasks

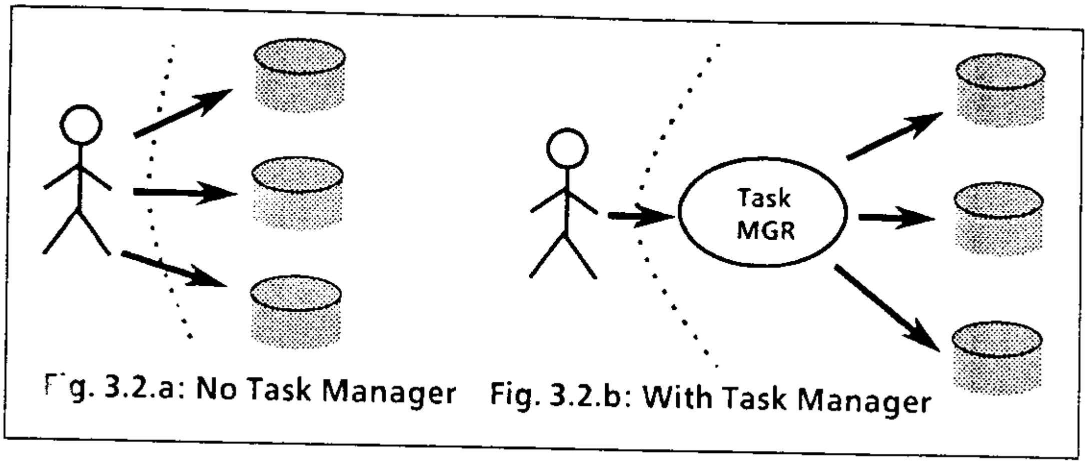


Fig. 3.2.a: No Task Manager Fig. 3.2.b: With Task Manager

may be assigned to the print Task Manager, filing - to the filing Task Manager etc. Thus the TM may be viewed as a front end to the servers managed by it and called **Front End Task Manager** or **FETM**. All requests must be handled by it prior to submission to the servers. The users supply it with the object and attributes. The FETM may be viewed as an autonomous service but users view it as an enhanced server. The FETM is shown in figure 3.3.

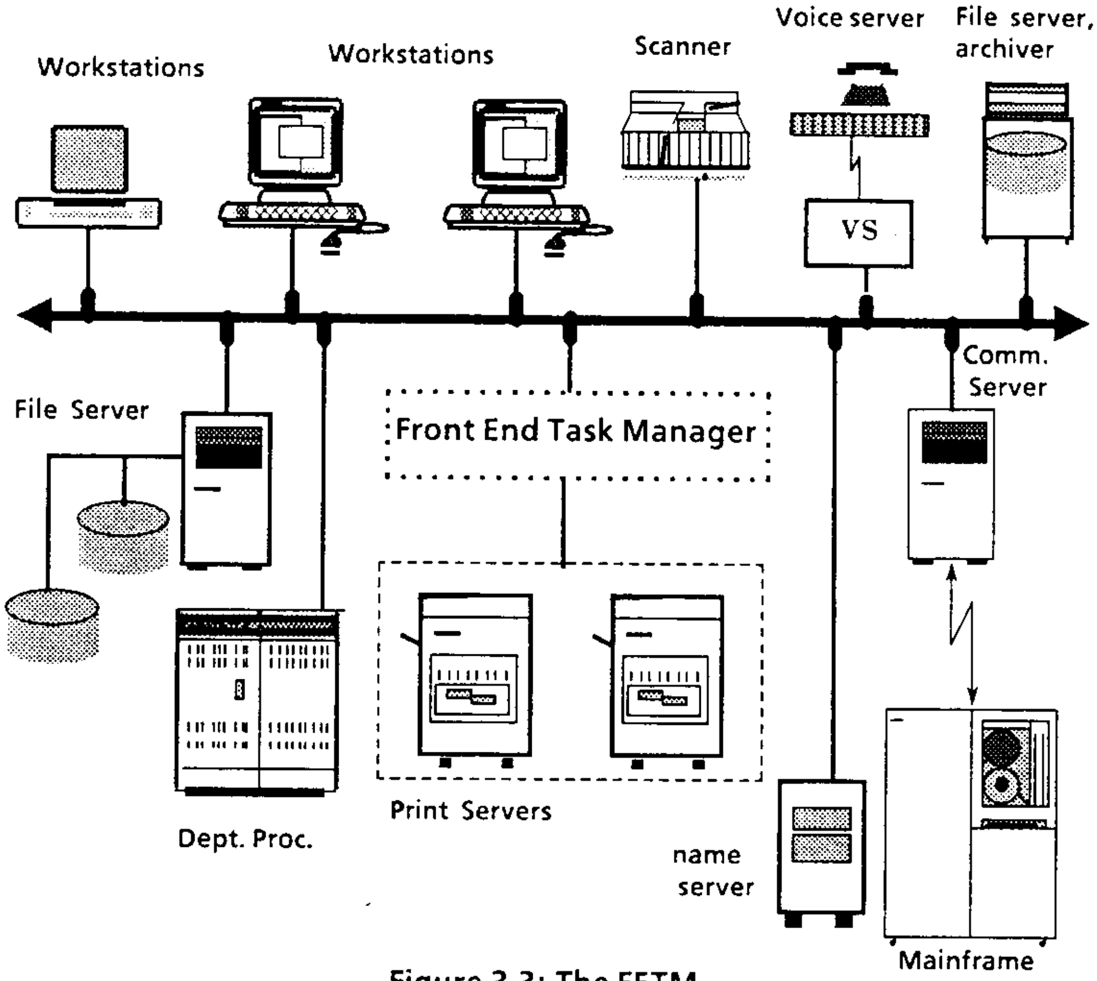


Figure 3.3: The FETM

The FETM concentrates on a unique set of tasks. However, another possibility is to manage a set of heterogeneous servers: a combination of scanners, file servers and printers, for example. Furthermore, this combination may be expanded to include archiving and other servers leading to generalizing the FETM concept to create the General Purpose Tasks Manager, or **GPTM** as depicted in figure 3.4. Note that the GPTM may be viewed as a collection of several FETMs. "TM" is used when no distinction between the FETM and the GPTM is needed.

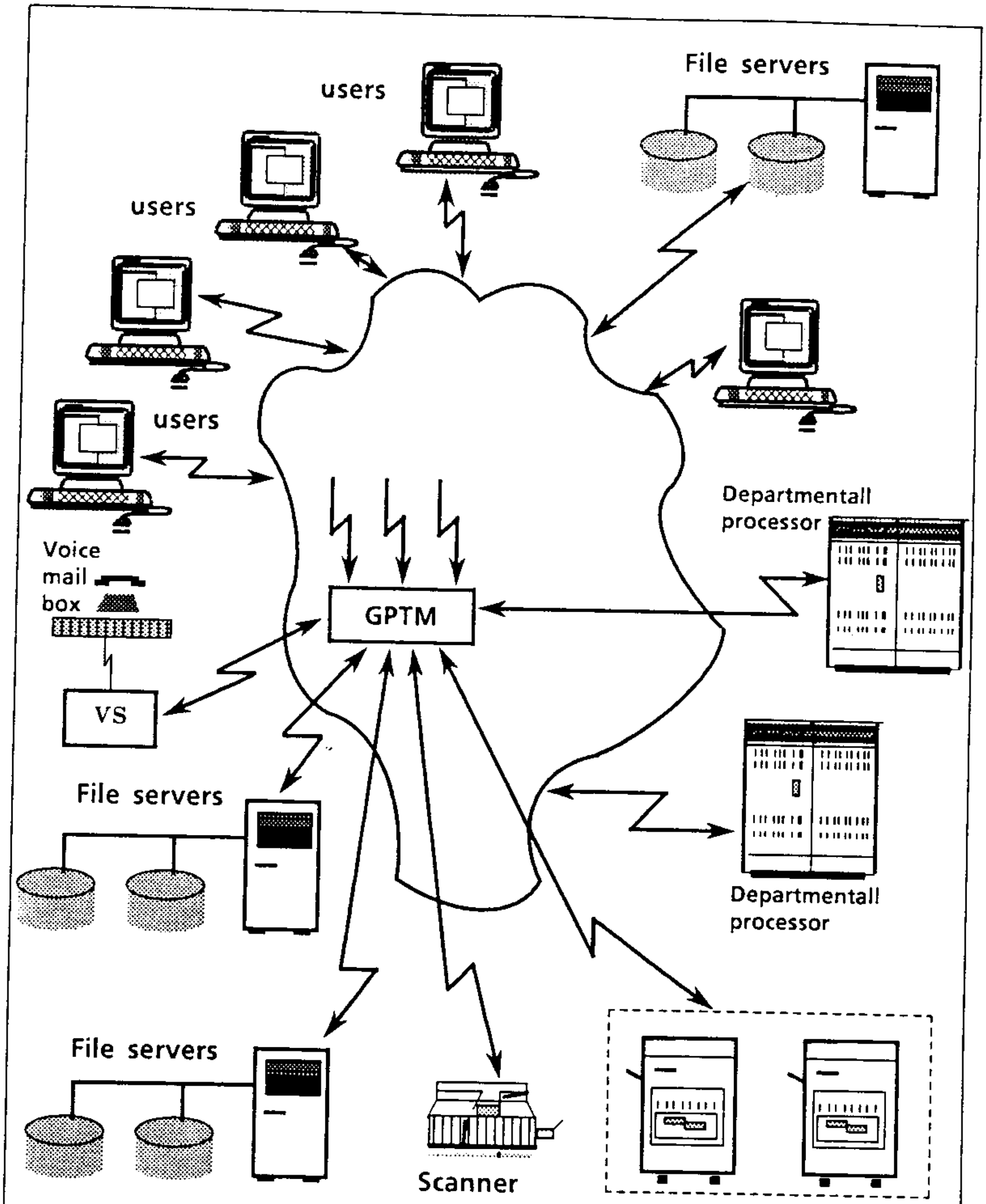


Figure 3.4: The GPTM

The TM controls the resources assigned to it. The introduction of the TM and its control over the servers raises the question of the availability of servers to direct user requests. Without the TMs, users transmit their requests to the "uncontrolled" servers who in turn execute them. With the TM, servers are not allowed to accept requests from any sender - in fact servers need to ascertain that all the requests come from their own TM. All other unauthorized requests should be ignored. Servers respond to their TM only.

The servers controlled by a TM are called **controlled** while the others - **public.** The mechanism enabling this feature is described later in this chapter. Controlled and public servers may be available in the same network. Thus, it is expected that servers should be able to be controlled by a TM and if needed, be directly available to the users.

In order to execute a job, the TM is provided with the object and attributes. However, the users may further direct the TM by providing it with user unique information. This information is provided in the **User Definition File (UDF)** which is similar to the mechanism available in Unix (.login, .cshrc etc.), Xerox Network System (.user) [Xer01] and in other systems.

The TM's **domain** is defined as the set of services available to and controlled by it. These services may or may not be local. The initial number and composition of the services assigned to a TM is referred to as the *initial set* or *initial domain*.

The initial set may be arbitrary and can be dynamically corrected by the TM. The TM may acquire and release servers on its own thus supporting *growth* and *shrinkage* phases. Hence the initial assignment may be small. The shrinkage and growth algorithms may be similar to existing algorithms like the Page Frequency Fault proposed by W. W. Chu and others, or merely consider server utilization. The algorithm may be selected based on its performance in a simulated environment. However, the TM may enter the growth phase once realizing it (a) cannot execute a pre-defined number of jobs per time unit, or (b) cannot provide a timely service. Once the 'grow' decision has been made, the next decision the TM needs to make is which server to add to its domain. It is not assumed that the system offers a pool of servers, one of which the TM may take thus the only option the TM has is to acquire a public server. A problem occurs when the TM cannot fulfill a very small number of service requests, a situation in which acquiring another server is unjustified. In this case, it may submit the service requests to a public server or to another TM which will, upon completion of the service, return the results back to the original TM. TMs submit jobs to other TMs as if the receiving TM was a server. In general, TMs communicate with each other as if the initiator is a TM and the recipient is a server. The TM receiving the job request from another TM is "exporting a service".

The shrinkage phase requires less decision making since once entering this phase, all is needed is to release a server which will be available to the public. Note that the shrinkage phase may be entered based on the server's utilization only. A different approach is not to allow the TM the capability of changing its domain but instead, to measure the servers utilization rate and rely on the **System Administrator** (SA), a human, to perform domain adjustments.

The TMs are capable of handling exceptions. Exceptions may be divided into two categories: (a) users assign wrong jobs to the FETM, and (b) TMs do not possess the capabilities needed to perform the job. Case (a) is trivial - the job will be rejected and the sender should be notified. In case (b), not performing the job is unacceptable thus the TM may submit it to a public server or to another TM as described earlier. Another possible solution is to re-partition the job into different tasks and execute them again (the original partitioning is no longer applicable due to a server failure, for instance).

Design and performance issues related to the TM are discussed later in this chapter. However, the topic of overlapping domains should be discussed here. The TM function may be centralized (referred to as CTM) or distributed (DTM). In the centralized case, several TMs may be available in the system each of which with its own domain. If necessary, TMs will exchange tasks as described earlier thus there is no need to share domains. In this case servers belong to one TM only.

Nevertheless, if the TM function is distributed, i.e. a TM per workstation, the TM domains must be shared. In this case, servers will belong in more than one domain and should only respond to the TMs controlling these domains. The shrinkage and growth algorithms are common to the CTM and DTM. But although the DTM offers many advantages, it poses a problem as well. Assume the TM's load balancing algorithm is dynamic i.e. it sends status messages to all of the servers, waits for them to respond with their queue length, and only then makes the task assignment. There might be a case in which the queue length at the task arrival time may not be the same as the queue length at the time the server responded to the status message not because there was a departure but because there were several arrivals from another TM. In short, how should several TMs concurrently balance loads between the controlled servers? The answers may be several:

(1) Restrict the load balancing algorithms to be static only.

(2) The optimistic solution: assume that arrival rates at the servers are low so that the problem will be minimal. And,

(3) The pessimistic solution: use locking.

The appropriate method to select one of the proposed solutions is by simulation.

The TM should not be mistakenly viewed as a centralized solution to the problem. Although all requests pass through it and it controls the servers, it is not the only one available and does not control all the servers. A set of fault recovery protocols is available to recover from a TM failure. In fact with the TM the overall system availability may increase due to the TM following up to verify successful completion of its tasks. Upon unsuccessful completion due to a failing server, it may redirect the job to a substitute.

3.3.2 TM's Resource Management Algorithms

TM resource management procedures are discussed in this section. In order to facilitate this discussion, figure 3.5 depicts some of the TM's logical steps. The *valid* step assures error free transmission and that the request is valid, i.e. a filing FETM received a filing request etc.

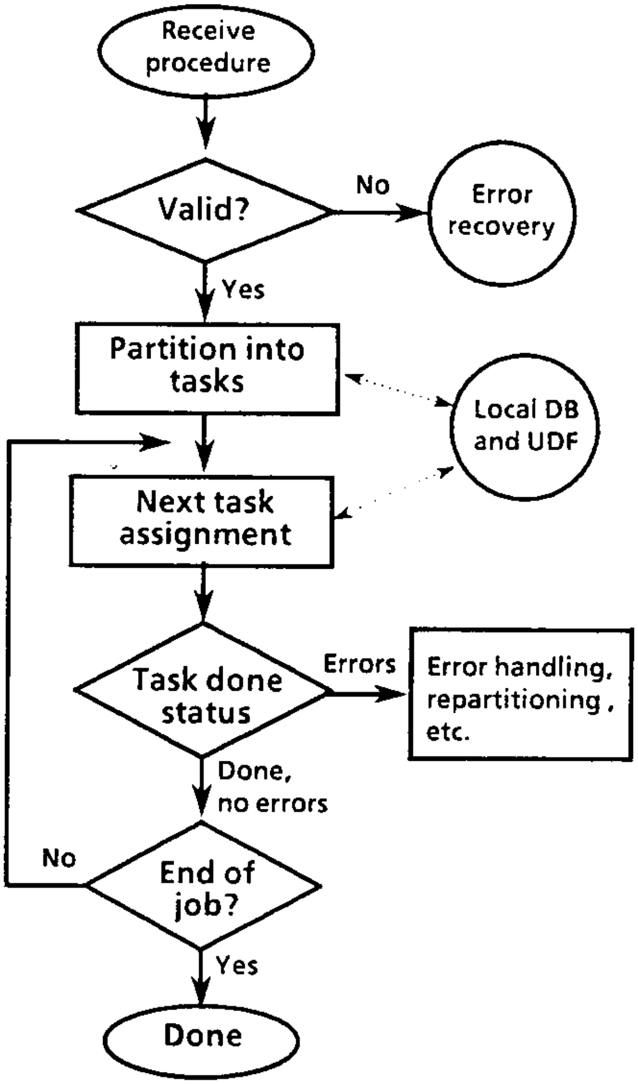


Fig.3.5: TM's operation sequencing

The next step is partitioning. Job partitioning and task allocation have been discussed in many papers including [Chu85]. It is believed that some tasks will require *remote file fetching* to fetch include files (also referred to as external references), a function which may not be supported by all servers. Therefore, prior to partitioning, remote files should be fetched. The attributes help task partitioning by revealing the steps needed e.g. printing a voice file necessitates an intermediate step of converting it into text before transmitting to the printer. Partitioning jobs into tasks considers the available servers and parallelism. Partitioning into tasks which cannot be performed by the available servers may prove useless.

In partitioning, the TM considers multi format issues. The object attributes describe the format and if necessary, the TM will need the help of a conversion service. Although most formats lend themselves nicely to conversion, some are difficult. Almost all user operations may be viewed as transfers: filing is a transfer from the source to the file server; retrieve - transfer from the file server to the home directory, etc. Since user transparency is required, the TM may have to deal with many new transfers: voice to file servers as well as printers, text documents to an audio server for listening, sections of moving video (images) may be rasterized and selectively printed, etc. It is assumed in this work that the TM has no control over the message routing or the communication network thus the transmission time becomes a function of the supporting network / system.

In some cases feature matching should be considered. As the system grows, servers with different features and capabilities may be added. To decrease waiting times and in anticipation for newly arriving jobs that might need a unique feature, f_a, the server offering it should primarily be handling jobs requiring this feature. The following strategy is implied:

> *If a job J_1 does not require feature f_a and can be equally assigned to servers S_1, S_2, ... S_n and only S_1 possesses f_a then J_1 should not be assigned to S_1. If not assigning J_1 to S_1 reduces the system utilization, the assignment should take place.*

Feature matching and load balancing may create a conflict without the second sentence in the rule above which has been added to cover for cases in which not assigning J_1 to S_1 may substantially disrupt load balancing.

Task assignment. The most important feature provided by the TM is the ability to perform load balancing satisfying a variety of constraints some of which are listed later in this section. In this work, though, the desire is to employ an algorithm that minimizes response time. This topic has been discussed in many papers and publications [dSe84] [Arv89] [Kle76]. Load balancing algorithms, also referred to as load sharing, are divided into **centralized** and **distributed** categories each of which may be **dynamic or static** as described in [Arv89]. Furthermore, load sharing may be **sender initiated** or **receiver initiated** as in [Eag86]. Centralized algorithms share the inherent characteristic of having a central focal point for determining the assignment of the next arrival to a server while in the distributed algorithms there is no single decision making node. The static algorithms perform load balancing based on a statically predefined procedure that does not consider the server's dynamic load at the time of assignment. On the other hand, dynamic algorithms consider server time sensitive parameters like calculated expected delay times based on its current queue length before deciding on the assignment [Arv89]. Sender versus receiver initiated algorithms deal with the topic of which entity should initiate load balancing. An extreme receiver initiated load sharing example is: whenever the receiver's input queue length equals zero, it will ask for a task.

All algorithms may implement fixed or variable thresholds for balancing. Fixed threshold implies that regardless of the servers load, the criterion upon which the decision will be made does not change in time. Finally, it is assumed that the algorithms are implemented at the user level which is consistent with the TM proposed implementation.

As described later, the TM itself may be centralized or distributed thus affecting the selection of the optimal load sharing algorithm. Nonetheless, there are no limitations in the TM restricting the selection.

In this work, though, a major network related assumption is made: transmission time from node to node is identical for all nodes. Thus the load balancing algorithm will not consider network transmission times in its decision making process. In addition, the TM may assign tasks to the workstation or servers back and forth many times until the job is done.

The following paragraphs describe five load balancing algorithms to be simulated in this work. However, this doesn't preclude the use of other algorithms. All of the algorithms are sender initiated and fixed threshold.

(1). Random or No Load Balancing.

Without the TM, it is assumed that the load is equally distributed among the servers. Each user will randomly pick one of many servers and assign the task to it. Therefore, this algorithm assigns tasks to servers with equal probabilities and regardless of their processing power or queue length. This algorithm is static. The advantage of this algorithm is its simplicity but its drawback is poor resultant response time relative to other ones.

(2). Server Capacity Relative.

This algorithm assigns tasks to servers based on knowing their processing power only, therefore it is static. It should perform equally well in centralized or distributed implementations and is easy to implement. It doesn't require additional communication overhead.

Given: a job to be assigned to the servers has arrived,
There are n servers, $m \leqq n$

C_m is server S_m's processing capacity in jobs/second.

Optimize: Response time

By: Static job allocation

Upon arrival of job J_k

Assign J_k to S_m with probability of $P_m = C_m / (\Sigma C_n$ for all **n**) and stop.

(3). Server Capacity Threshold

This algorithm assigns tasks to servers based on knowing their processing power and dynamically querying their queue length size therefore it is dynamic.

Given: a job to be assigned to the servers has arrived,

There are **n** servers, $m \leqq n$

C_m is server S_m's processing capacity in jobs/second.

Optimize: Response time

By: Dynamic job allocation

Upon arrival of job J_k

```
1. For each queue: get total queue length $l_m$
2. For each queue: calculate server load $L_m$
   $L_m = l_m / C_m$
3. Find $S_m$ satisfying $\min(L_m)$ for all n
4. If only one queue found in step 3, assign $J_k$ to this queue and
   stop, otherwise skip this step.
```

5. If no one single queue found in step (3), assign J_k to S_m with probability of $P_m = C_m / (\Sigma C_n$ for all **n**) and stop.

A distributed version of this algorithm should consider the problem mentioned earlier: Assume source **A** is busy performing steps 2 and 3 above. Possible arrivals from another source may nullify the queue length sizes transmitted to **A** in step 1. In our simulation the optimistic solution has been implemented. The capacity t/h algorithm should provide better response times but is more complicated to implement and imposes communication overhead compared with algorithm number 2. Note that step 5 is identical to algorithm #2.

(4). Server Queue Overflow.

This algorithm assigns tasks to servers without knowing their processing power but based on their queue length size therefore it is dynamic.

<u>Given</u>: a job to be assigned to the servers has arrived,
There are **n** servers, $m \leqq n$
C_m is server S_m's processing capacity in jobs/second.

<u>Optimize</u>: Response time

<u>By</u>: Dynamic job allocation

Randomly select a server, call it S_1. Randomly select a maximum queue length threshold **TOR**. Upon arrival of job J_k

1. Set **m** = 1.

2. Get S_m's total queue length l_m
3. If $l_m < TOR$ then assign J_k to S_m and stop. Otherwise go to step 4.
4. Set $m = m + 1$. If $M > n$ assign J_k randomly and stop. Otherwise go to step 2.

This algorithm offers the advantage of not having to know the servers capacity. It also requires less communication than algorithm 3. A potential problem with it occurs when the selection of S_1 happens to be the slowest server and the last one, S_n, is the fastest. Therefore algorithm 5 is proposed. The effect of various maximum queue sizes, **TOR**, should be addressed.

(5). Optimized Server Queue Overflow.

This algorithm is identical to algorithm 4 with the following exception: S_1 will be the fastest server, S_2 is the second fast . . . S_n is the slowest. The advantage of this algorithm over the previous one is obvious. The penalty is the need to know the server's processing speed which, in our opinion, is not a serious penalty for the potential response time improvement.

Load balancing is not the only criterion upon which the tasks are assigned to servers. As mentioned before, feature matching and the UDF are considered as well. The UDF takes the highest priority in the decision making.

A local database supports the TM's decision making. To perform task assignments, the TM should have information about the servers in its domain. Since this information is needed repeatedly, once the TM inquires the servers, it should store the server attributes locally. The TM is responsible for database updates and is the only network entity with access to it. As domains expand or shrink, the TM will update this database.

The procedure in figure 3.5 will be performed repeatedly until the job terminates at which point a Done message may be transmitted to the user. In each step the TM may collect statistics to facilitate optimizing its operation and to report it to the SA. For instance, if the TM detects that a remote file is repeatedly needed, it may store a copy of it locally rather than spend time retrieving it over and over.

The TM will need to include additional real life considerations as described in the following examples:

Minimum waiting time per job. The TM may disregard additional overhead and inefficient use of resources in order to get a job executed in minimum time. Example: a high priority job which needs yellow paper will be submitted knowing that it will require the operator to change paper from white to yellow and most probably from yellow to white later. The job will be finished fast but it imposes overhead and thus affects other jobs in the system.

Maximum resource utilization. Keeps the servers busy even if it delays some jobs which need attention. Example: a job which needs yellow paper will be delayed until there are no more jobs waiting or until there are enough jobs that need yellow paper.

Minimum cost per job. A job may be executed at the manager's discretion in order to save money. The manager has the authority to executed it at the time and location leading to minimum cost based on information provided in the job attributes.

3.4 Queries and Responses

The queries and responses can be divided into two groups: those between the user and the manager and those between the manager and the services. Users interact with the TM only.

The TM can interrogate the servers in its domain as well as interrogating other TMs as if they were servers. It can inquire about their current load, free disk space (if any), their capabilities and options. The TM will have to dynamically update its database in order to make resource allocation decisions.

3.4.1 Queries Between the TM and the Users:

Queries between the TM and the clients may include:

1. Capabilities and capability sets. The manager needs to report its capabilities as if it were a 'super server'. It will report the capabilities of its controlled servers (including dynamic attributes) in **sets** of which only one may be selected. If there are additional capabilities due to the existence of private services, they will be added to each set. The sets are independent of each other thus allowing series or parallel task assignments.
2. Status_ID. The manager needs to report the status of the specific jobID: it's location in the server's queue (if still active) or its done status (done, done with errors, aborted).

   The query "Status_ID" must be responded to with up to date information since the manager can delete jobs from a server's queue and immediately ask for StatusID.

3. Cancel_ID. Delete the job identified by its ID. Users may delete their own jobs only.
4. Prioritize_ID. The job should be assigned the new priority, if priorities implemented.

3.4.2 Queries Between the TM and the Services:

Queries between the manager and the services may include the ones listed in section 3.4.1 and the following:

1. I-am-your-mgr. The service will respond with a positive or negative acknowledgment. See figure 3.12.
2. I-am-not-your-manager-anymore. The manager informs the service that it will not be controlled any more. The service responds with positive acknowledge. The TM may not send this instruction until after it has received acknowledges that all of the jobs submitted to that service have been processed. The System Administrator can also issue this instruction.
3. General poll ("how busy"). This poll will be responded to with total queue length and possibly amount of free disk space.
4. List all jobs. The System Administrator will be able to get the list.
5. List all services. The System Administrator will be able to list all the services the TM controls.

6. Add-a-service. Each service needs to know whether the jobs it receives has been authorized by the manager. Only services within the manager's domain may transfer jobs to other services in that domain. They do so after being told by the manager. This instruction notifies the controlled services that they may get jobs from a newly added service. All other jobs should be rejected.
7. Remove-a-service. The compliment of add-a-service.
8. List manager. This is equivalent to "who is your manager".

3.5 Protocols

Systems similar to XNS, as described in [Xer01][Xer03], require the users to 'manually' direct their tasks. In case a user receives a document represented in a foreign standard or format he will be required to convert it before reviewing. He will need to follow the sequence of events as depicted in Fig. 3.6. In this example, the document is sent to the converter and back to the user. Once it has been read, the user may store it on the file server.

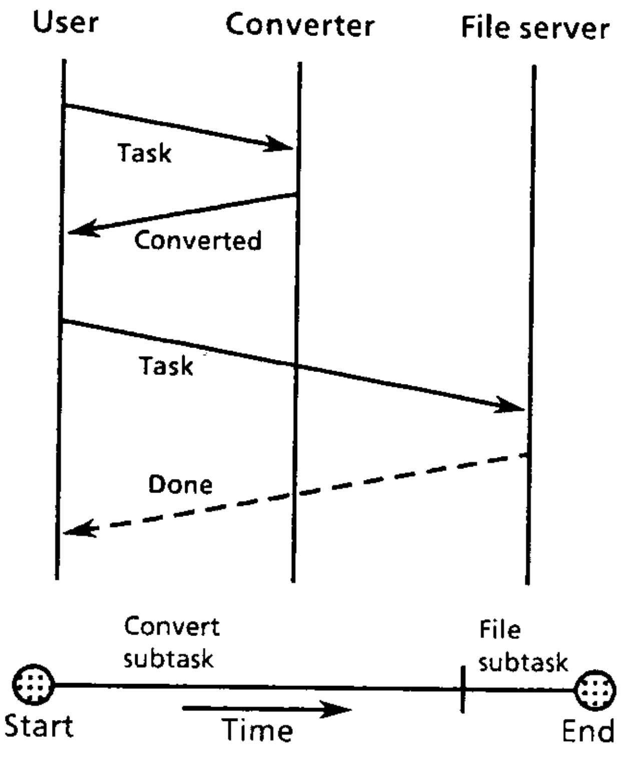


Figure 3.6: A transaction without a TM.

In general, all jobs may be divided into tasks, some have to be performed locally and others may be performed better on a designated server.

Depending upon the specific case, the server may or may not need to transmit the results to the sender (the TM or the user). Its output may be temporarily stored locally and then sent to the next destination. Some servers do not generate electronic output: printers. A client who wants to submit a job will interact with the TM. The TM will be supplied with the object and attributes, if available, in order to make decisions regarding the processes to be run during the job's life, their order and final disposition or output. Figure 3.7 represents control and data flow. It shows a general activity sequence as a function of services needed and time. The basic operation of the manager is depicted and that pattern holds for additional service-request/service-done procedures.

A major architectural decision has been to view the TM as responsible for all actions following receipt of a job. It acts on behalf of the users if a need arises to redirect jobs or resolve conflicts or make any needed decision. A second decision has been that the TM should not prepare a list of processes to invoke immediately following the reception of a job but rather partition it, select the **first** process to be invoked and decide on the next step once the previous one has been completed. The advantage of this approach is that the task assignments will be done in real time thus taking advantage of the most up-to-date information (how-busy) received from the servers. Another decision was that users may submit jobs even if they do not possess all of the job attributes. The TM will have to try and resolve this problem by extracting the attributes from the object itself, assign default values or even send the job to a server better equipped to perform this task.

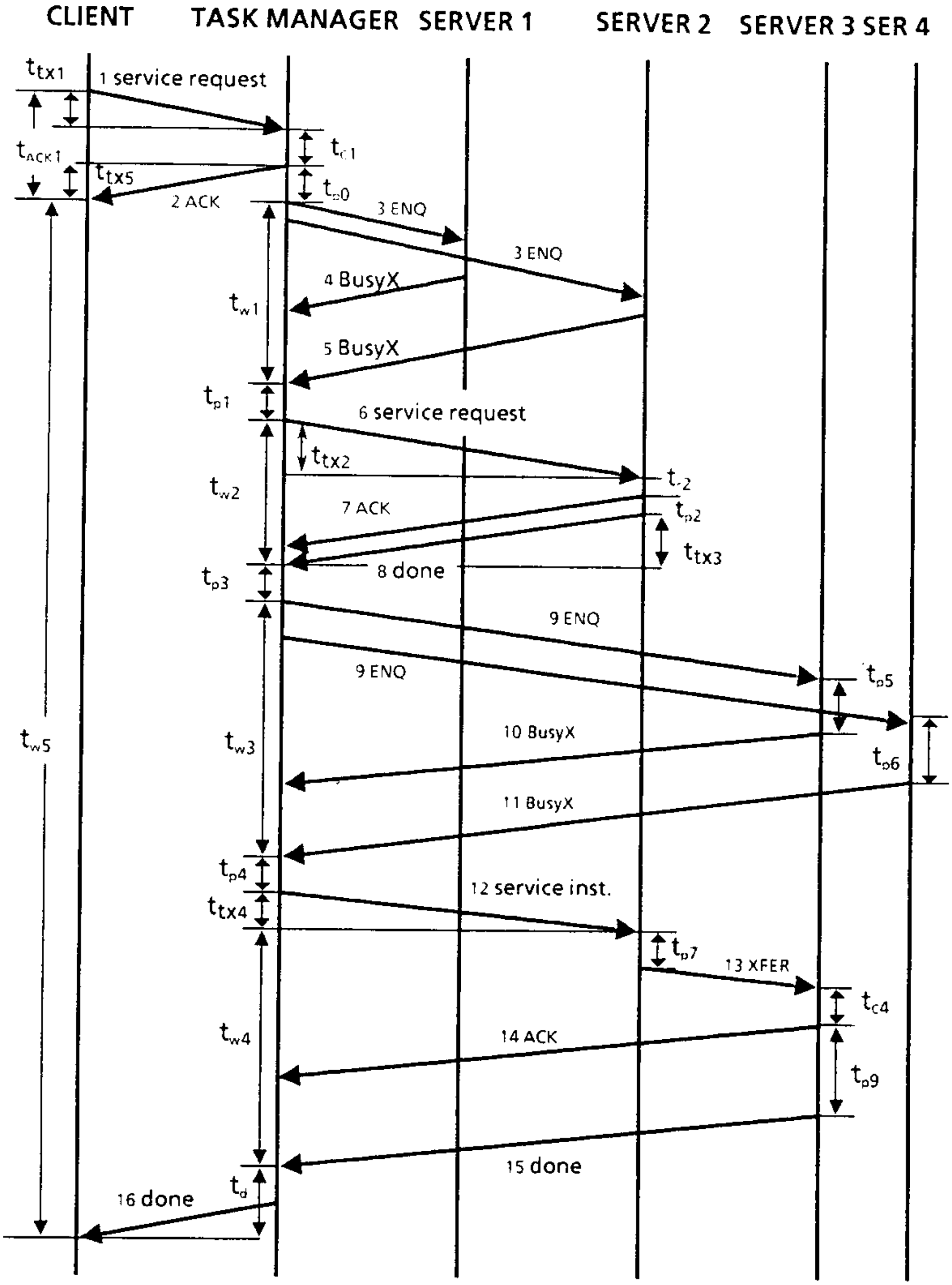


Fig. 3.7 Data and control sequences

Fig. 3.7 step 1: The client transmits the service request to the TM.

The user's request should supply the manager with the object and attributes. Both are necessary in order to facilitate its decision making with minimum overhead. The attributes, if available, will be transmitted in this "service request" phase. Attributes submitted may be insufficient. The object itself may be attached or, alternatively, may reside in the network and the request should have a pointer to it (filename plus pathname). The TM will always expect to get information about the task itself like "include" files, file type, length and external routines in the "attributes" section so that it will not have to re-scan the object. This information is needed in order to perform feature matching, load balancing and assure that the include files are local or that they can be retrieved prior to job execution. It should also supply the direct job related attributes (in the case of printing like number of copies, media, and finishing; in case of voice: compression algorithms etc.)

The user may not have the above information at hand. In this case he will send the direct task related attributes plus the [object] or [object name] and the manager will have to get the attributes needed by itself, a process that will require one more step. Only one object is associated with a request, but the object itself may solely be made of include files.

There are several related issues regarding the above:

1.1 The object creator, who knows everything about the file should provide the necessary attributes.

1.2 Where should the attributes be stored? The object-related attributes may be stored in the object itself. Dynamic, or direct job related, attributes do not need to be stored anywhere. Attributes may be missing or specified in a contradictory way. This leads to the need to comply with some sort of attribute priorities, i.e. (highest to lowest): System Administrator, job submitted, taken from the object, server default.

1.3 For the cases where the user does not provide the necessary information to the TM, the TM should have a service that can do that. This information will be transmitted back to the manager. If the manager has this service, it should be available to the clients too, so that they will be able to provide the information to the manager with the next request related to that object. This means that the service may be public and the manager as well as the clients will be able to use it. Note that in the ideal case the manager will not need to use it since the users will.

Fig. 3.7 step 2: Task Manager acknowledges reception.

This optional acknowledgment includes an ID and indicates that the request has been accepted. Note that the user expects to receive this "ACK" within a pre defined time frame for fault tolerance as discussed in the following section. Upon ACK he knows that the manager is up and running and that he (the user) may proceed as follows:

2.1 If he submitted the attributes and the [object OR a pointer to the object, which points to a file server!] and did not indicate that he wants to be notified when the task is done, then he will proceed to his next task which may be unrelated to this request.

2.2 If he submitted the attributes and the [object OR a pointer to the object, which points to a file server!] and indicated that he wants to be notified when the task is done, then he may proceed to his next task but expect to be notified.

Design issues outside the scope of this work include: The pointer may point to file servers only. Clients may not point to themselves or other clients. Should ACK be restricted to "the job has been received with no checksum errors"?

Fig. 3.7 step 3: TM polls the selected services ("how busy").

At this point, the TM has acknowledged reception of the job. Next is job execution by:

(a) partitioning the job to tasks in such a way that the tasks can be assigned to servers in the TM's domain, see section 3.3 and

(b) assign the tasks.

The manager processes the information transmitted to it via the BusyX message. If it doesn't have the needed information to manage the job (print servers may need fonts, for example), it will need to use a service to supply it with the information prior to any other activity.

The local database and the UDF will be used in selecting the next service. The local data base is composed of data collected in the recent past from previous inquiries about server capabilities. There might be two or more server candidates. (The task may also be assigned back to the originating site). In order to identify one of them, the manager issues a "how busy/ENQ" poll to the services and waits for their response. The type of poll (general or specific) will affect the traffic on the network but has no importance here. The polling sequence is of no importance either.

"How-busy/ENQ" denotes the number of jobs in the input queue and is needed only for dynamic load balancing algorithms. Ideally, the response should be in time units [Arv89], however, this cannot easily and accurately be done. It is impractical to estimate execution time of submitted jobs without probing into them first, a step which will add extra load. In addition, the reported data may be incorrect due to the possibility of queue changes due to arrivals of higher priority jobs in case priorities are used. There is no unique "how-busy" poll. There is a general poll by the manager, which will be replied to by units of "how-busy" and units of free disk spooling space. See "Queries and Responses".

Fig. 3.7 steps 4 and 5: Services respond to the TM.

The response denotes how busy the services are and also implies that they are "alive". It may also include free disk space.

Note that there should be no pending jobs to be transmitted to these services. This assumption must hold in order to ascertain that the service will be in the same state (or better) by the time it actually receives the message from the source. This means that the input queue to the service, sampled at the time of "how busy", already includes all the jobs destined to the service from all other sources in the manager's domain.

Fig. 3.7 step 6: TM sends attributes plus object to service #2.

The TM uses the procedures described in section 3.3, Resource Management, and decides to use service #2 (load balancing and other considerations). It sends the object plus needed information to the destination.

Fig. 3.7 step 7: Service #2 to TM: acknowledge.

Service #2 optionally acknowledges receipt by sending a message (with the ID) to the manager who in turn clears its "receipt pending" switch. This procedure assures that there are no cases where the manager interrogates the service before the service received all the files destined to it (possibly from another party). Responding to "how-busy/ACK" queries before receiving all the jobs from the various sources may yield incorrect reporting and thus should be responded to by "pending".

Fig. 3.7 step 8: Service #2 to TM: done.

Service #2 notifies the manager that the job is done by providing the following information: error free or problems detected; output filename and size. The file may or may not reside on the server's disk. In this work it is assumed the file is resident, otherwise, a pathname should also be provided.

Fig. 3.7 step 9: TM polls server#3 and server#4 (how-busy).

The manager decides on the next type of service. It identifies two servers that can do it equally well and wants to find out which one is less busy. If needed, the above procedure will be repeated recursively.

Fig. 3.7 steps 10 and 11: Servers respond with units of "how busy".
The servers respond as described earlier.

Fig. 3.7 step 12: TM instructs service #2 to transmit its output to server#3.
It does so since server#3 is less busy. It expects an acknowledge from it.

Fig. 3.7 step 13: Service #2 transmits the attributes and object to server#3.
Upon transmission, service #2 deletes the input and output files from its disk. At this layer of the protocols it is assumed that the destination has a correct copy of the transmitted data i.e. the output of service #2 will not be needed again due to transmission errors. Further, it is required that if the selected server cannot fetch remote files, these files will be filed on its disk prior to this transaction.

Fig. 3.7 step 14: Server#3 to TM: acknowledge.

Server#3 acknowledges the TM that it has received the job. The manager clears its corresponding switch.

Fig. 3.7 step 15: Server#2 to TM: done.

Server#2 notifies the manager that the task has been accomplished with or without errors. Output disposition is done based on the attributes. It deletes the job from its local memory only upon successful completion and only if it was not instructed to store it.

Fig. 3.7 step 16: Optional: TM notifies user that the task has been completed.

This option should be selected/not-selected by the user. It is very useful in assuring faultless operation. It will default to no message.

The procedure described above assumes servers do not need to remote fetch. In case there is a need to remotely fetch but the servers aren't able to, the TM should pre-fetch the files prior to submission to the servers. The include file names appear in the attribute part thus assisting the TM. If the servers can remote fetch then the sequence described in figure 3.8 should be employed. Step 1 (the request) specifies where the object resides. The manager acknowledges the client (step 2) and submits the request (object and attributes) to server 1 in step 3. Server 1 acknowledges reception and proceeds to fetch the needed file (from outside). It notifies the manager that

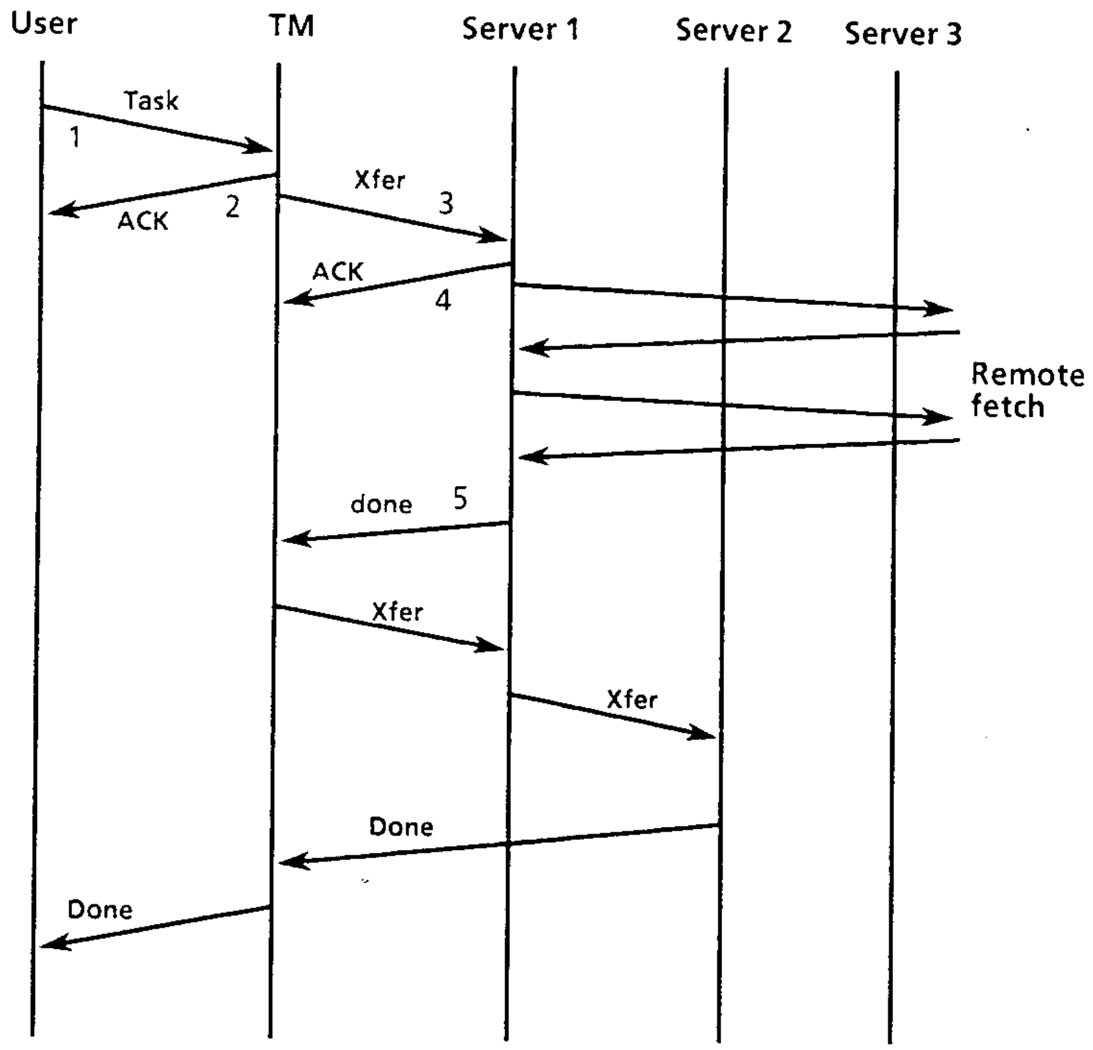


Figure 3.8: Remote fetch

it's done in step 5 and from this point on, the sequence is identical to the one described in Figure 3.7.

Figure 3.9 describes the case of a simple print request. The client requests service and gets acknowledged. The manager does not poll the printer and in the XFER message sends the attributes and object to the printer. Since the client did not request to be notified, the manager does not take any further action following the done message from the printer.

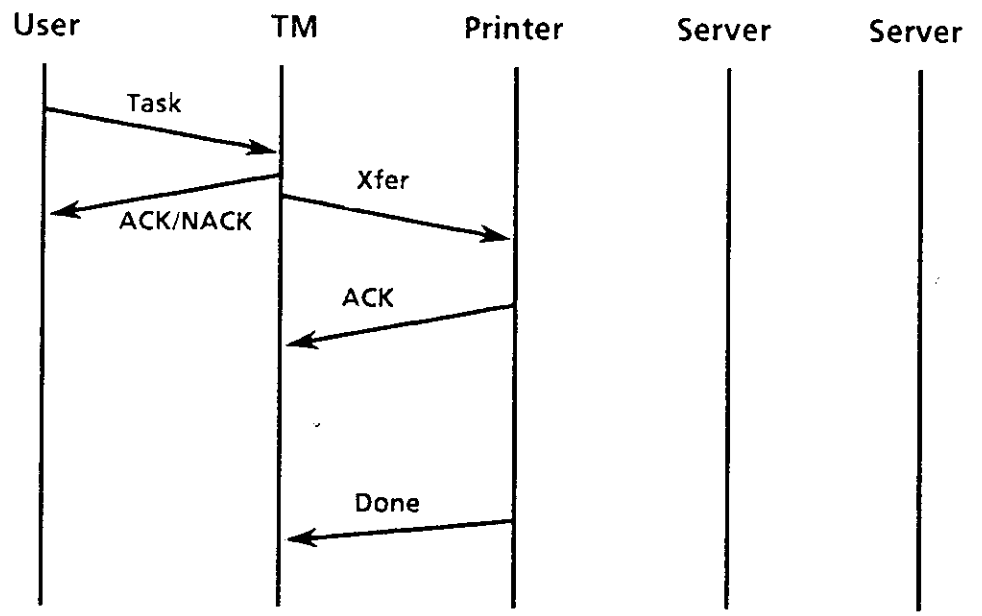


Figure 3.9: Simple Xfer, no poll, no notification

Figure 3.10 depicts the protocol sequence in case execution errors occur. Once the TM receives the task it polls the available servers and XFERs the task to the selected server, in this case server 1 which starts processing leading to detecting an application oriented error (out of disk space, a vector processor in the compute server is malfunctioning, stack overflow etc). Upon receiving the Error message, the TM submits the task to the second best server capable of performing the task which is server 2.

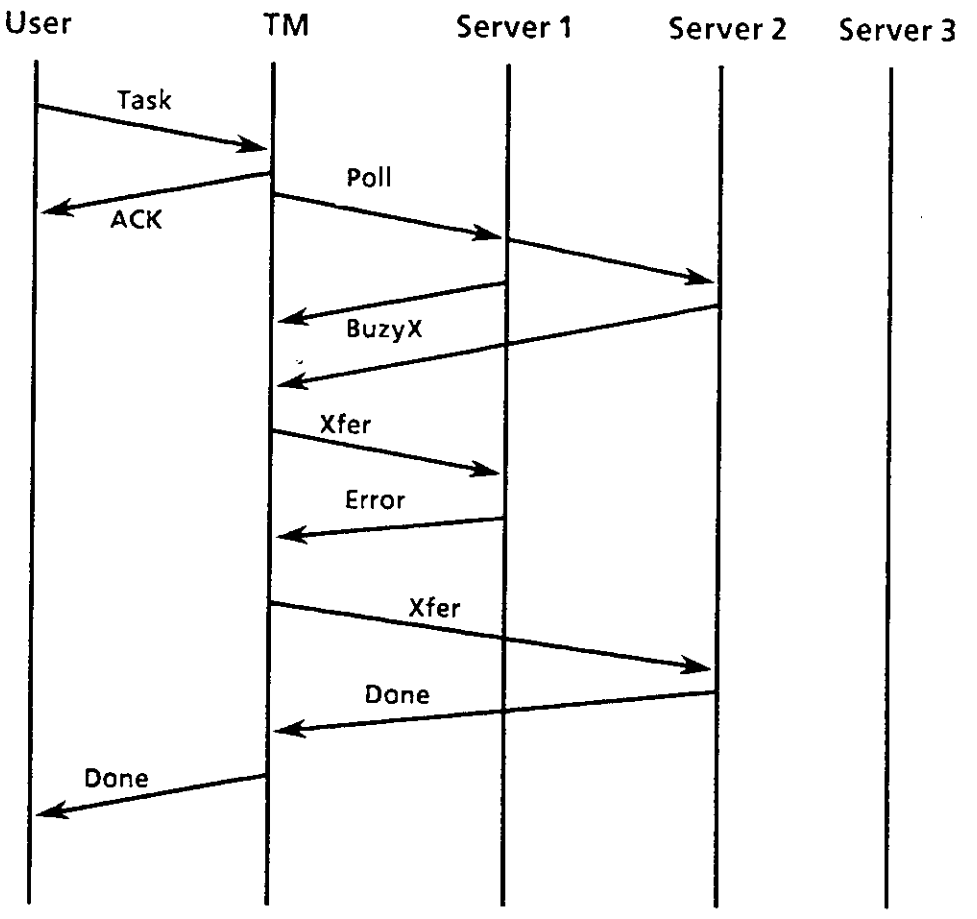


Figure 3.10: The case of execution errors

Figure 3.11 depicts handling user inquiries. It is clear that all inquiries pass through the TM: users are unaware of the servers supporting the TM nor do they have the knowledge where the job is being executed.

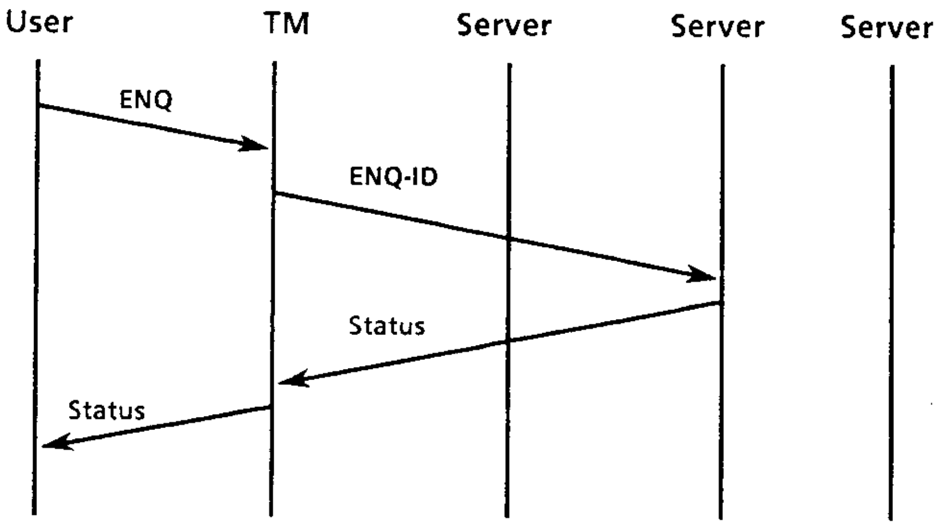


Figure 3.11: Handling a user inquiry

In figure 3.12 cases A and B depict manager initiated requests, one of which is I-am-your-manager. The server, granted it may be controlled, proceeds to validate the request since it has no knowledge who may control it and there is also a need to authenticate the request. Based on the authentication results, the server responds with either ACK or NACK. Case B is simpler since the server is not programmed to be controlled by a TM, i.e. it is available to the general public only.

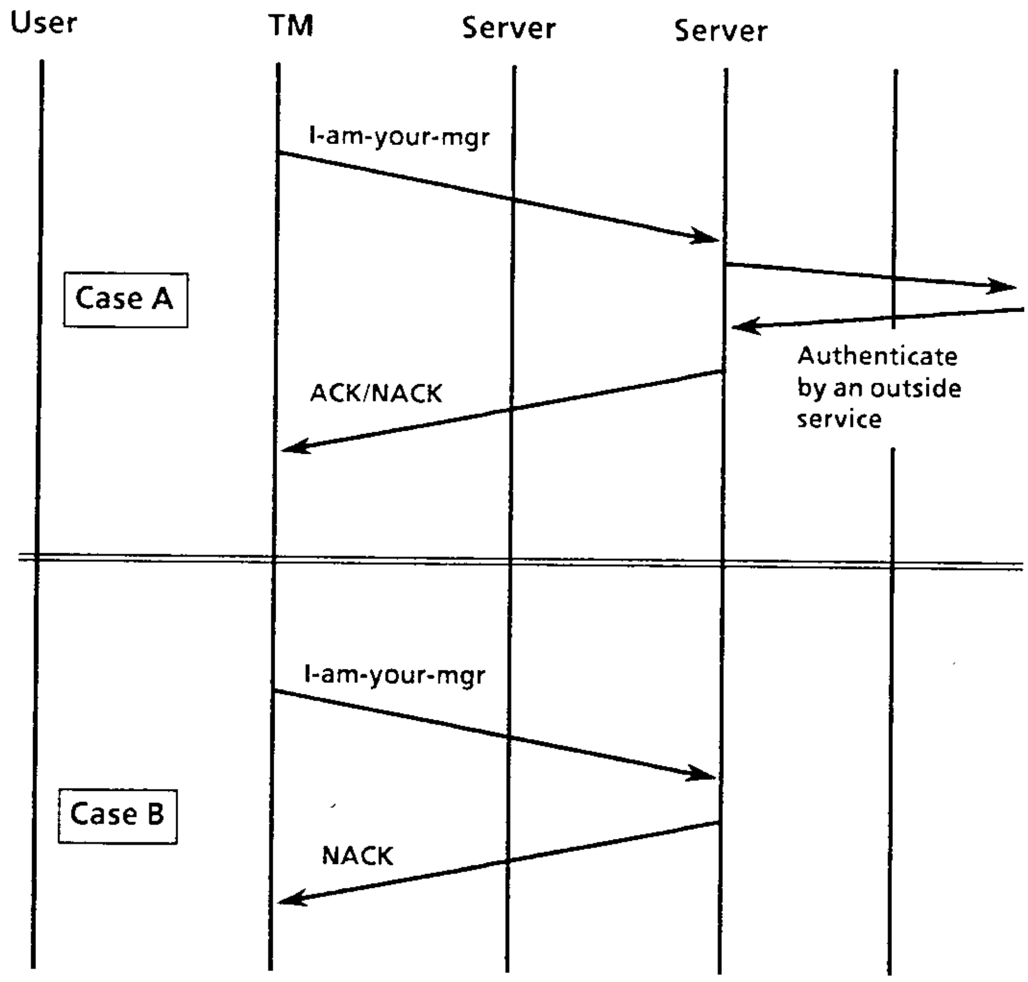


Figure 3.12: Server control requests (A and B)

3.6 Fault Tolerance Considerations.

TM as well as server failures should be considered. Some implementations may elect to rely on the fault detection and recovery procedures provided by the lower level protocols thus in fact leaving TM failures to be resolved by the users. Operative TMs will resolve server failures as discussed in the previous section (as with server errors) thu only TM failures will be discussed.

3.6.1 User Detected Failures

In systems not providing TM failure detection the users have to detect TM failures and notify the pre-selected secondary TM. No "are you alive" messages are exchanged between the users and TMs. No TM history files are being backed up either. Upon failure detection, users notify the secondary TM which has been statically defined; the secondary TM verifies failure and becomes primary. All other users will follow this procedure and experience delay till they communicate with the secondary TM.

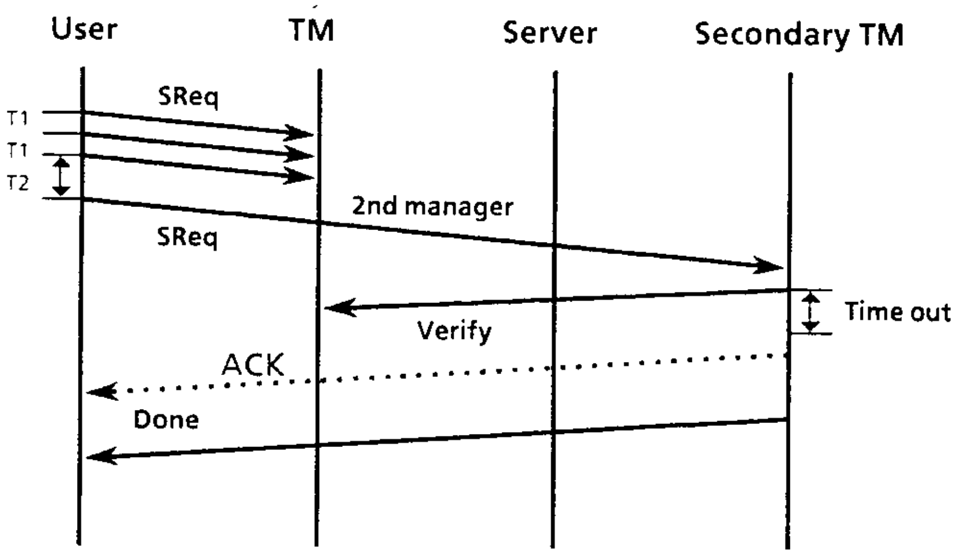


Figure 3.13: TM fails before submission

Figure 3.13 depicts the event sequence when the TM fails before a user requests service. Users may send Service Request (SReq) messages to the TM till they suspect a TM failure. It takes T2 seconds to prepare the SReq message destined to the secondary TM. Once the secondary TM receives the request, it verifies that the primary TM is down, executes the requests and responds with ACK. The verification step, however, is needed only following the first SReq message transmitted to the 2nd TM. If a TM fails at any other time the user has to verify job completion. If incomplete or unknown, users have to resubmit. It's their responsibility to prevent duplication, if any and an optional ACK from the secondary TM may be sent to the user acknowledging receipt of the job. The ACK may in fact be substituted by "busy" units, if desired.

The procedure described above is unattractive. Users must keep a copy of each task submitted to the TM and follow up once the task has been completed. In addition they will have to take the necessary steps to recover from TM malfunctions which in fact burden them with an extra task considered overhead. This approach is inefficient since the users will have to restart a job from its first processing step and not take advantage of possible processing that may have been done prior to the TM failing (assuming TM failed after the job has been submitted). Furthermore, temporary files may reside on various disks without the users being aware of them: the orphan problem. There is also a need to "release" servers who have been managed by the TM. (Note that this last problem could be resolved by implementing "are you alive" messages between the servers and the TM. Upon failure, the

TM would not respond to the server initiated message thus indicating a failure prompting unilateral server action to be released from the TM). Another drawback of this scheme is the possibility of overloading the TM with status messages processing. These drawbacks overpower the advantage of simplicity.

3.6.2 Aided Detected Failures

As noted, the procedure described above is deficient in several key points: lack of transparency; users need to detect the failure and notify a **statically** designated secondary TM; the TM may create a bottleneck; multiple simultaneous failures are not supported unless more than one secondary TM are statically assigned; each user has to independently detect the failure. Some of these drawbacks may be alleviated by relying on a "safe and secure" server. This server already exists in most networks: a name server or clearinghouse [Pet88] [Opp83]. The name server will always supply the users with the address of the currently assigned TM which may be changed dynamically.

There is no need for the name server to detect failures. If the TM fails or **even** if it has been decided to shift the TM function from one node to another the old TM will not respond thus the users detecting it will enquire the name server for the correct TM address. If it has been changed, the name server will respond with the new address but if it fails, an "are you alive" message will be sent to ascertain failure, then an assignment message will be sent by the name server to the newly selected TM and the user will be provided with the new TM name. The "are you alive" sequence will be performed only once to verify failure. This protocol is depicted in figure 3.14 below.

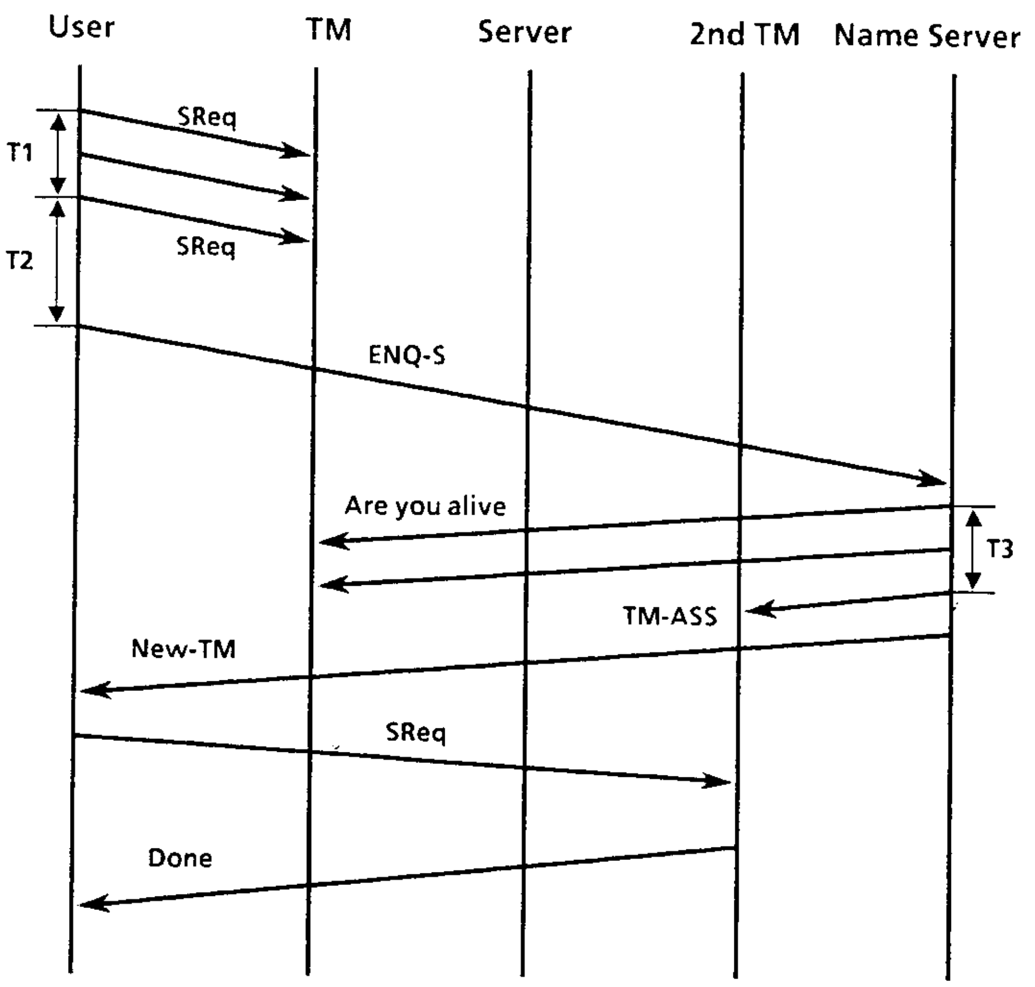


Figure 3.14: TM failure recovery.

This approach can tolerate multiple TM failures. Users aren't aware of TM failures and the overhead imposed is low. There is no need for any control messages to detect problems. The secondary TM assignment can be done in real time. A drawback of this procedure is the need for the safe and secure server. Its failure may prove very problematic to the network. Even if history files are available, they are not backed up thus a TM failure will result in losing these files as well. The name server performing this function should not overload it since failure rates are very low and the overhead imposed is minimal. However, as with the approach described in 3.6.1 the old TM domain may be lost. This may be resolved by keeping redundant domain and history files.

3.6.3 Transparent Failure Detection

This approach requires an initial assignment of secondary TM. 'I am alive' messages are transmitted by the primary TM to the 2nd TM as has been proposed in [An85] However, this algorithm can be improved as follows:

- No secondary TM is statically identified.
- "Are you alive" messages will be transmitted by the TM to the name server not to a secondary TM.
- Upon TM failure, the name server will time out.

From this point the algorithm is similar to 3.6.2.

Note that in the protocols, if a TM fails and there is no history file identifying the failing domain and its controlled servers, the servers, upon timeout, may send 'are you alive' messages to the TM. If it does not reply, then they will change status from "controlled" to "public". If primary/secondary TM approach is adopted, the secondary TM (now primary) will have to select a new secondary TM.

3.7 Design and Performance Considerations

At this point in the research data and control transfers, the TM concepts and the need for resource management algorithms have been discussed. In this section the discussion is centered around two TM cases and a set of design and performance criteria to be considered leading to a specific and detailed XNS study/simulation. The two cases are: (1) the *distributed TM (DTM)* as depicted in figure 3.15 and (2) the *centralized TM (CTM)* as depicted in figure 3.16 . In case 1, each workstation is assigned a TM whereas in case 2 the TM is centralized and supports multiple workstations. Each distributed TM supports its domain which is initially specified by the individual users or the System Administrator based on their knowledge of the network but the domain may change to accommodate dynamic needs for resources as

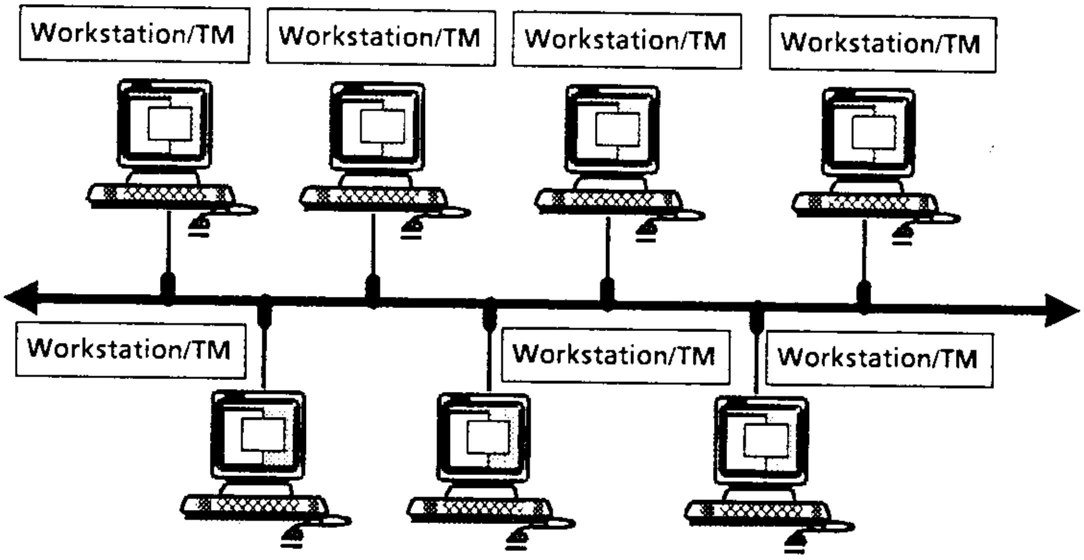


Fig. 3.15: Distributed TMs

described in section 3.3. The network servers do not belong to a specific TM therefore no TMs "own" services. Contrasted with case 2, the TM in case 1 is less prone to creating a bottleneck since it shares the processor with other tasks and may be allotted larger time slices if needed. As discussed in section 3, the TM will recover from server failures thus increasing reliability. DTM failures imply that the workstation has failed thus affecting one user only.

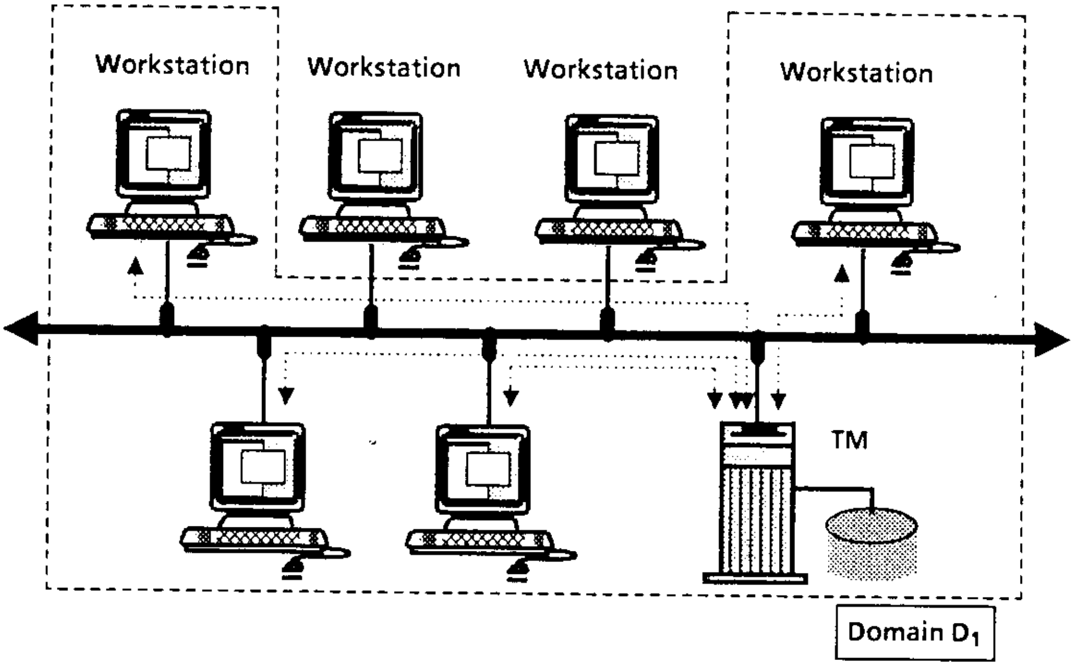


Fig. 3.16: Centralized TM

In case 2 - the CTM function has been moved to a dedicated server common to a set of users - the first design consideration is how many TMs should a network possess. This is coupled with another question: how many users should be allocated to a TM so that it will not create a bottleneck. Assuming that the network utilization is very low, less than 5%, the major parameter to consider is the TM server's throughput. For the entire user population in the network, the number of TMs should provide for "adequate" TM response time which may be obtained by simulation. The initial TM domain is set by the System Administrator and may be changed by the TM itself as needed. The domain is specified in accordance with the type and number of users: a researcher domain may include several computing machines while draftsmen may need more plotters. Nevertheless, the CTM is a potential bottleneck if its throughput is not high enough and furthermore, a failing CTM should be replaced expeditiously (see Fault Tolerant protocols in section 3.6). A potential problem with the centralized TM is the need to rapidly access the UDFs. Since each UDF is controlled by its user, it would be logical to store UDFs at the workstations. This situation will necessitate the CTM to remotely access these files thus increasing the network traffic, increasing network load and yielding higher delays. A potential solution is to store UDFs at the CTM site while maintaining user rights to modify them. As mentioned earlier, in both cases the TM will handle server failures and depending upon the principal design, TMs may also keep copies of jobs submitted by the users thus relieving them from fault tolerant issues.

In addition to the design considerations discussed above, two more topics should be addressed: (a) statistics collection and (b) relation to the ISO 7 layer architecture. Regarding issue (a), the CTM supports better statistics collection since all transactions pass through a single point - the CTM. Although the distributed TM also collects statistics, it lacks the advantage of accessing a common location for statistics retrieval. For issue (b) it is assumed that the TM is Application level oriented and belongs in level 7 of the ISO architecture. The advantages of this approach are: network independence - the TM does not rely on a particular network feature, it does not require or impose any standard additions/changes, and it may be customized for specific requirements. Considering it as a lower level feature, possibly level 4, has the advantages of faster performance since there is no need to pass data through the entire 7 levels and due to lower implementation overhead (number of procedure calls).

The impact of adding the TM on the system **performance** should be evaluated. The single most important performance parameter the users are sensitive to is the job **response time,** also referred to as the *open loop response time* since jobs arrive to the network, get processed, and terminate.

The following parameters / tradeoffs and their cumulative effects are evaluated in chapter 4:

(1) Arrival rates.

It is expected that arrival rates will vary: start at an inter arrival time of 100 seconds and increase to over 400 seconds. The effect of different arrival rates on the response time should be studied.

(2) TM load.

Adding the TM functionality will (a) require additional processing capacity from the resource on which the TM is running, or (b) the TM will contend for the existing capacity. Assuming case (b), the effect of various TM loads on the response time should be understood.

(3) Number of workstations.

Given a TM implementation, the effect of varying the number of workstations on the response time should be addressed.

(4) Workstation processing power.

The response time sensitivity to workstation processing power variances should be studied. This is especially important in case the TM is allowed to assign tasks to the originating workstations (referred to as cooperative workstations). This parameter will be varied assuming that workstation processing power may only increase.

(5) Load balancing variants.

Adding load balancing has the advantage of improved response times due to improving resource utilization but dynamic algorithms (compared with static) also impose extra communication overhead. An issue to be addressed is whether the extra overhead associated with a selected dynamic algorithm does not nullify its potential response time improvement compared with a selected static algorithm. Load balancing algorithms are discussed in chapter 3.

(6) Server processing power.

Many different servers with different processing speeds may be available. The server's processing speeds should be changed to ascertain that the TM can effectively handle a range of server speeds.

(7) Job size and its processing time.

Various job sizes as well as their processing time requirements are indicative of job complexity and their effect on response times should be studied.

(8) User think times.

Tasks performed by the users will be done by the TM. Performance improvements depend on how long the users need to think at the workstation therefore this parameter should be varied and effect on the response time recorded.

(9) Centralized versus Decentralized TM.

As described earlier, the TM may be centralized or distributed. Each configuration offers its advantages but the effect of centralizing the TM on response time should be simulated.

Some of these variables can be fully controlled: server and workstation processing power, load balancing algorithms, etc. while others' have to be considered: user think times, IATs etc. Nevertheless, they all affect server utilization and communication overhead and may create bottlenecks. See chapter 4.

Understanding these topics will enable a meaningful performance tradeoffs discussion as done in chapter 4. The effect of varying these parameters should be quantified. Otherwise, the discussion may only be intuitive and lack needed support, therefore, simulation will be used to make conclusions.

Finally, an observation follows: the CTM is more susceptible to creating a bottleneck at the CTM site. If the workstations are cooperative, i.e. the TM is allowed to assign jobs back to them, the amount of network traffic may increase substantially thus creating another potential bottleneck - the network. In this case, assigning the TM to the individual workstations may yield less traffic since DTMs may assign tasks locally thus eliminating the need to transfer files. In addition, assuming (a) the TM's requirement for processing power is low, and (b) the workstations are not loaded, a TM per station may get more CPU time and perform faster than a dedicated station for a centralized TM serving several users. Nevertheless, in both cases the TM, a machine, will perform the tasks currently done by the users faster than them. The simulation in this research will try to quantify the performance improvement.

3.8 Related Topics

(a) To support third party data transfers, services will need to verify that the jobs they get from other servers have been authorized by their manager. The add-a-service and remove-a-service procedure will be used as described earlier.

(b) Services know whether they are controlled by a manager. If they are, they know who their manager is and respond only to him (the who-is-your-manager query should be excluded). They should obey the I-am-your-manager instruction only after verifying that the requestor can be a manager. The authentication protocol (or similar if not XNS) will be able to support this requirement. This is done in order to prevent clients from monopolizing printers.

(c) Secondary storage space. The manager will be able to store files for future usage. This may be done using existing protocols (Filing).

(d) Statistics. Since the manager controls the system it can gather valuable statistics regarding frequency of usage, average load, down time and billing information. System planners may need to monitor overall system utilization, a function that can be performed by the TM.

4.0 Case Study

This section studies the TM performance in a real life Xerox Network System (XNS or NS) environment. The purpose of the study is to assess the possible performance improvement offered by the TM if implemented as an application layer feature. As discussed earlier, the TM should offer many advantages including shorter response times due to its associated resource management -and other- procedures. However, the addition of any utility contending for processing power or generating extra network traffic may lead to increasing queuing delays thus nullifying any performance improvements. This concern has to be addressed and resolved. Therefore a decision to use simulation was made. The simulation helps to quantify parameters like network response times and resource utilization but cannot show improved transparency, for example.

One of the objectives was to develop a simulator suitable for any environment facilitating the distinction of the TM from the communication network and the workstations. In this specific research, XNS parameters like network speed and workstation processing power have been used. This approach allows us to assess performance **with** and **without** the TM enabling an objective comparison and assessment of the performance improvement attributed to the TM. Therefore, the XNS network has been simulated separately and results from this simulation are used in the TM application level simulation. The set of issues investigated in the XNS as well as the TM simulations is described later in this chapter.

This chapter also provides a very short description of the Xerox Network System as needed to support this work. Detailed description may be found in [Xer01] [Xer02] [Xer03] [Xer04] [Xer05]. The protocols specified in XNS are an open ended set of packet transport protocols used uniformly across a variety of communication media, processors/servers and office applications all of which vary from installation to installation and from time to time. Higher level protocols built on top of the Internet Transport Protocols provide the necessary mechanism to transfer data and identify data type.

The intent of this chapter is to verify, substantiate and quantify the response time improvements achieved by implementing the TM. Furthermore, an important aspect of this work is the fact that the TM concept is being simulated based on a real life network in a real life environment rather than an abstract theoretical scenario. It evaluates the effects of changing network and workstation parameters on response time to which users are very sensitive. It also tests load balancing algorithms in a real life environment. Section 4.4 covers the TM simulation results, analysis and conclusions revealing the effectiveness of the TM.

4.1 XNS Overview

XNS is a packet oriented network architecture with four layers, the first three of which are called *Internet*. A packet contains control and data where the data may range from 0 to 540 bytes as explained later. Internet packets are routed through the internetwork as datagrams via store and forward elements called *Internetwork Routing System.* The source and destination addresses are encapsulated in the datagram packets leading to treating each one of the packets independently. The internet gives its best effort to deliver an internet packet but it cannot guarantee duplication free transmission nor sequencing.

As with other networks, XNS also employs a layered architecture as depicted in figure 4.1. Each layer defines a type field that is interpreted by the next higher layer, providing a bridge between the two layers. The layers are defined as follows:

Level zero. There is a need to physically transmit data from one point to another. This level is highly dependent on the particular transmission medium involved. There may be different level zero protocols and some of them may contain internal levels. Although XNS allows for a set of physical level implementations, XNS as implemented today is heavily dependent upon the Ethernet as defined in the "Blue Book" IEEE 802.3 [Eth82].

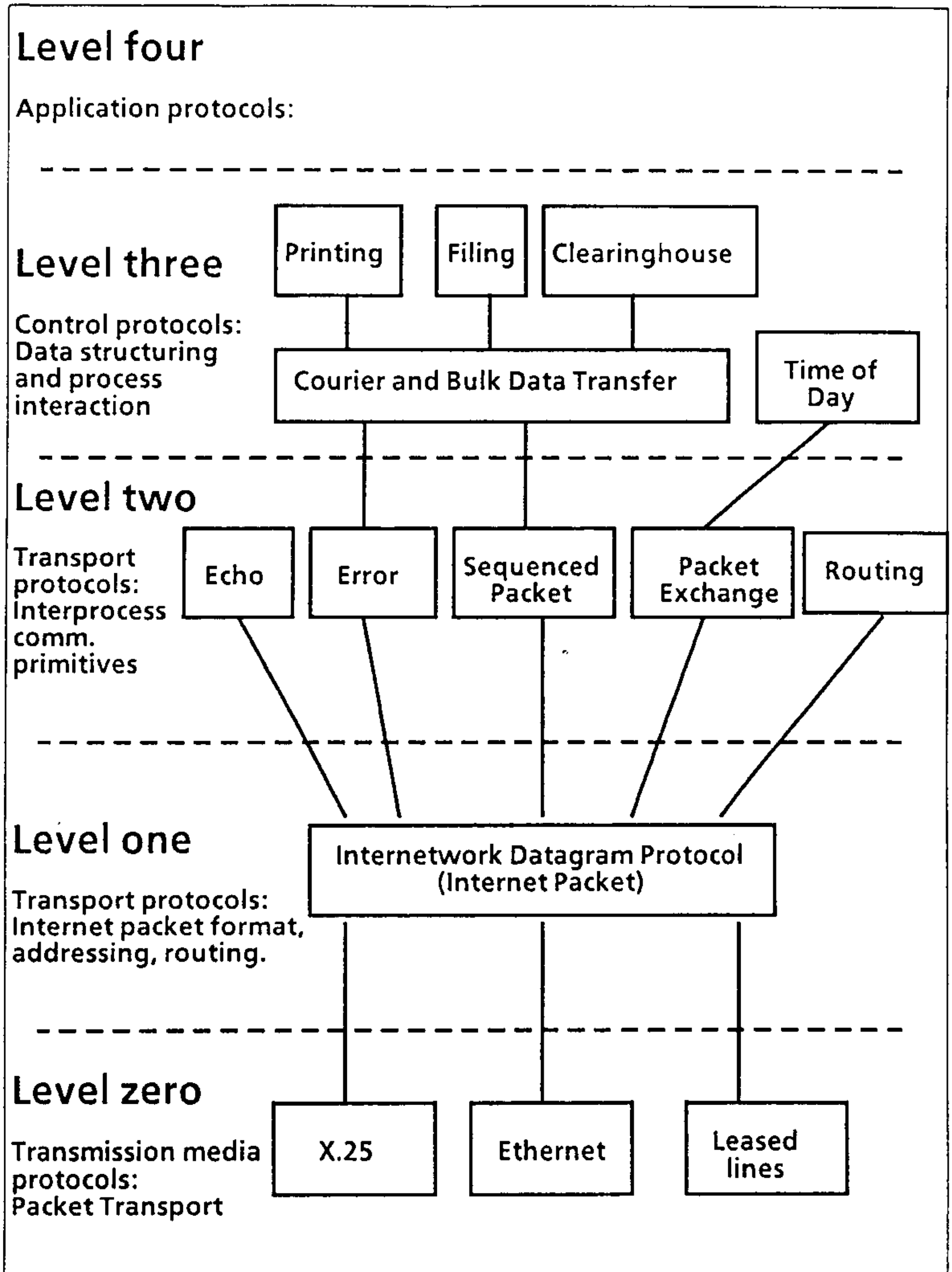

Level four
Application protocols:
Level three
Control protocols: Data structuring and process interaction
Printing
Filing
Clearinghouse
Courier and Bulk Data Transfer
Time of Day
Level two
Transport protocols: Interprocess comm. primitives
Echo
Error
Sequenced Packet
Packet Exchange
Routing
Level one
Transport protocols: Internet packet format, addressing, routing.
Internetwork Datagram Protocol (Internet Packet)
Level zero
Transmission media protocols: Packet Transport
X.25
Ethernet
Leased lines

Figure 4.1: XNS layers

Level one. This level is primarily responsible for routing and is called Internet Datagram Protocol. There is only one level one protocol and it defines the internet packet and rules for its delivery as a datagram (addressing and routing).

Level two. This level contains transport protocols. This involves retransmission, sequencing, duplicate suppression, and flow control. The set of protocols specified in this level provide for a multitude of requirements from higher level protocols such as high priority transmission, error control, routing information etc. Alternate implementations are allowed. XNS provides Echo, Error, Sequenced Packet, Packet Exchange and Routing protocols.

Level three. This level has less to do with communication and more with the content of data and the control of resources thus leading to naming it the control protocols. It includes the Courier and Bulk Data Transfer protocols [Xer05] as well as Printing [Xer04] , Filing [Xer03], Clearinghouse and Time of Day protocols and others.

Courier: The Courier protocol specifies the manner in which an active system element (workstation, the TM, etc.) invokes operations provided by a server or a passive element. Courier itself is a layered protocol using remote procedure calls to initiate connectivity and exchange data. "Layer one of Courier, the lowest layer, defines a block stream which can carry blocks of arbitrary binary data between system elements. Block streams are defined in terms of the connection abstraction of the Sequenced Packet Protocol. Layer two defines an object stream capable of carrying structured data between system elements. Object streams are defined in terms of the block stream abstraction of layer one. Layer three defines a message stream capable of carrying service requests (call messages) and replies (for example, return or abort messages) between system elements. Message streams are defined in terms of the object stream abstraction of layer two" [Xer05].

Printing: The Printing protocol is a set of procedures to set the printer state to the desired one (media type) and transmit the object file to the print server. It also supports status requests [Xer04].

Filing: The filing protocol specifies file contents formats, attributes, handles and controls (concurrency controls), directories, creating, accessing and deleting files. It uses Courier and Bulk Data Transfer to transfer data, if a transfer is requested [Xer03].

4.2 Current XNS Submission and Management

As described earlier in section 1, the submission process may require several steps. Once the job has been submitted, there may be a need to follow up in order to report status, redirect or cancel the job.

XNS supports a set of protocols at the application layer including the Printing Protocol and the Filing Protocol. The printing protocol allows submission of a selected file to a pre-selected printer. In addition, it also allows status reporting back from the printer to the user upon request. Nonetheless, none of the other submission or management requirements as listed earlier are being addressed. With some enhancements, this protocol may serve a key component in the overall solution.

The filing protocol [Xer03] may be viewed more as a set of remote procedure calls supporting file transfers. It has the needed support for file attributes and allows for the standard file manipulation procedures. In comparison, the printing protocol is more task oriented and may be modified to address a multitude of tasks including voice, still and moving images.

The submission process is merely the user submitting his task to a server he selected. There is no load balancing done. Feature matching is static: users have to gather system knowledge ahead of time in order to facilitate it, there is no capability to redirect a task nor cancel it from the user's terminal or workstation (once it has been submitted). In summary, only step {f} in the submission process (actual submission by file transfer) is supported thus requiring the users to have system knowledge hence reducing transparency. Job management, as defined in chapter 3, is limited to status reporting: no load balancing and no resource management done.

4.3 Delay, Parameter and Terms Definitions

This section defines the parameters and terms used. It covers delays as depicted in figure 4.3 and variables used in the simulation.

Figure 4.2 simplistically depicts the sequence of transfers in order to print a document residing on the user's workstation. It shows that only one step (submit) is needed for printing. The job is done asynchronously and at any time, the user may request status. We would like to develop an expression describing the elements contributing to the average response time R_{tm} **with** and **without** the TM. Without the TM this expression may be very straightforward. Comparing it with the expression developed for the TM may prove meaningless unless user thinking times (and others) are introduced. Therefore, Fig. 3.7 which is been duplicated here as figure 4.3 will be used for comparison with appropriate values used in both cases.

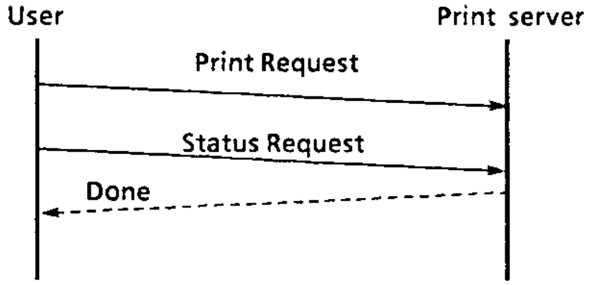


Fig. 4.2: Basic print request

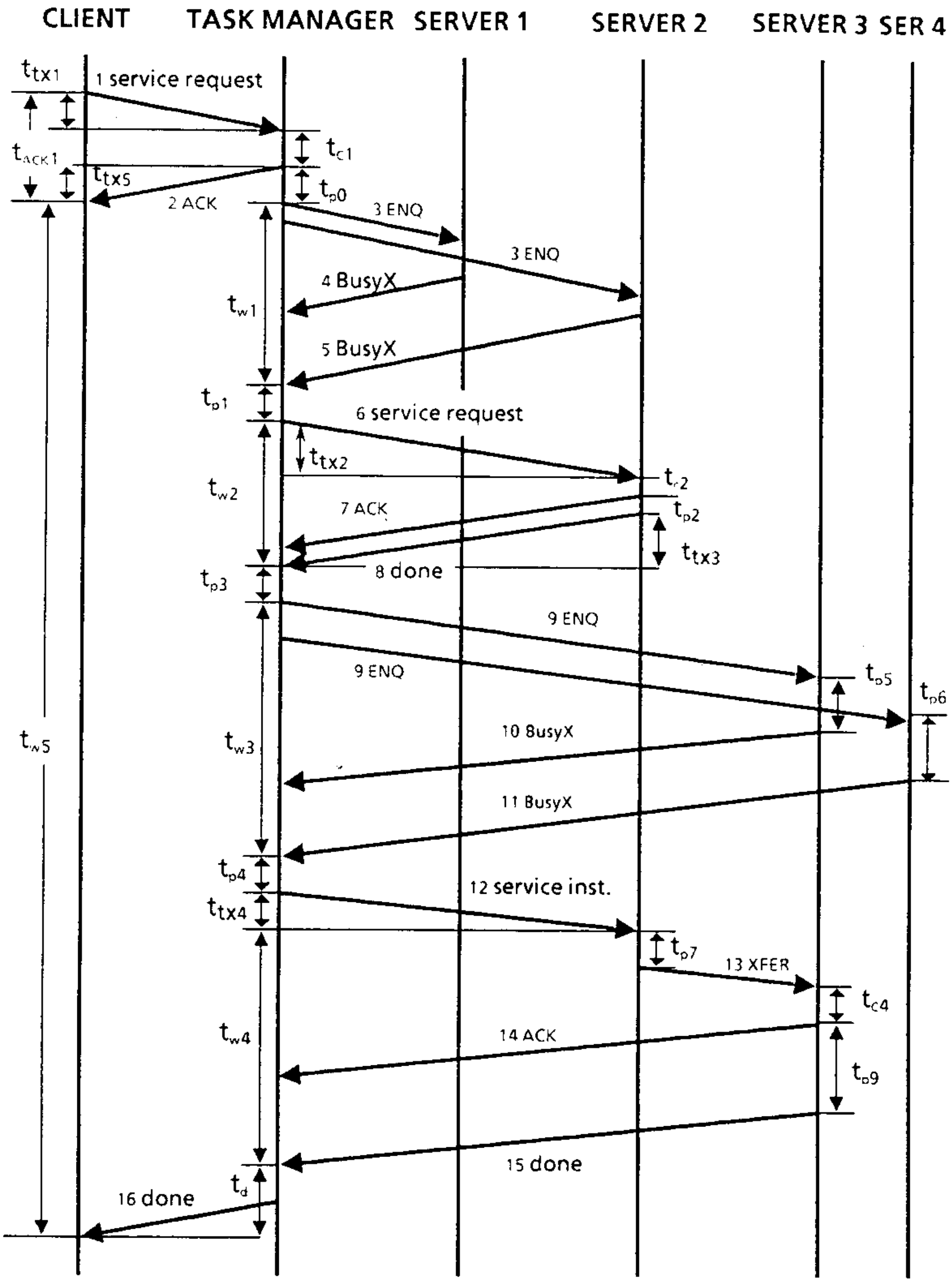


Fig. 4.3: Data and control sequences

t_{tx1} = Service request transmission time: client to the TM. This is the total time it takes the network to transmit the request, handle transmission errors, suppress duplicates and and handle Ethernet collisions. Note that user thinking time precedes this time and is not a part of it.

t_{c1} = Time for the TM to validate the request and prepare for the ACK message. It does not include any process time needed to parse or partition the job.

t_{tx5} = ACK transmission time.

t_{ACK1} = $t_{tx1} + t_{c1} + t_{tx5}$. This is the total ACK wait time.

t_{p0} = Processing, parsing, partitioning time plus local database inquiry times at the TM site.

t_{w1} = TM waiting time for both inquiries to be transmitted, processed at the servers and received through the communication network.

t_{c1} = time to validate request type and completeness before sending ACK.

t_{p1} = Processing time at the TM site for both BusyX messages. It also includes the time needed to prepare the service request message.

t_{tx2} = Service request transmission time.

t_{c2} = Similar to t_{c1}

t_{p2} = Execution time at server 2 including the Done message prepare time.

t_{tx3} = "Done" transmission time.

t_{w2} = service turn-around time at server2 = $t_{tx2} + t_{c2} + t_{p2} + t_{tx3}$

t_{p3} = Decision making time at the TM needed to select next step(s).

t_{w3} = similar to t_{w1}

t_{p4} = Time to process steps 10 and 11 and prepare the Service Instructions message. Similar to t_{p1}

t_{tx4} = Service instructions transmit time.

t_{p7} = Time to process the service instructions message and prepare for the XFER in step 13.

t_{c4} = Similar to t_{c2}

t_{p9} = Processing time at server 3 and time needed to prepare the Done message.

t_d = The sum of processing time at the TM and transmission time.

t_{w5} = Time between last ACK received and the DONE message.

- R_{tm} **[seconds]: Response time** (also referred to as open loop response time) is defined as the average life time of a job in the system and equals $R_{tm} = t_{ACK1} + t_{w5}$.

- **IAT [seconds]: Inter arrival time** is defined as the reciprocal of inter arrival rate.

- **Utilization** is defined as the time the resource is busy divided by the total busy and idle times.

- **Network delay [seconds]**: is defined as the time a packet spends in the network. The packet may be transmitted over the network more than once.

- **Throughput [jobs/sec]**: is defined as the average number of departures per time unit.
- Load balancing names are defined in 3.3.
- **think [seconds]**: is defined as the time NS users spend thinking at the workstation prior to submitting the job. See 4.4.1(d).
- **thinkz** [**seconds**]: in NS, once the job has been submitted, it is directed to a server. When done, the user may have to think for *thinkz* seconds about possible subsequent processing.
- **mess [seconds]**: is defined as the job processing time at the workstation. If the distributed TM is implemented, *mess* includes its processing time requirements as well.
- **con [seconds]**: control/status processing time at the workstation.
- XNS and NS are synonymous.
- "Servers" are distinct from workstations.

R_{tm} [with TM] = $t_{ACK1} + t_{w5}$ =

$$t_{tx1} + t_{c1} + t_{p0} + t_{w1} + t_{p1} + t_{w2} + t_{p3} + t_{w3} + t_{p4} + t_{tx4} + t_{w4} + t_d$$

In a more generalized form the total time may be expressed as follows:

R_{tm}[with TM] =

{ Submit time = t_{tx1} } +

{ Total TM enquiry times: $\Sigma\,[t_{w1}, t_{w3} \ldots]$ } +

{ Total server process times: $\Sigma\,[t_{w2}, t_{w4} \ldots]$ } +

{ Total TM process and transmit times: $\Sigma\,[t_{p0}, t_{p1}, t_{p3}, t_{p4} \ldots, t_{tx4}, \ldots]$ }

Thus R_{tm} may take the form:

R_{tm}[with TM] =

{Submit} + (this is step 1 in fig. 4.3)

N_1x {TM enquiry} + (these are steps 3, 4 and 5 etc.)

N_1 x {transmit and server process} + (these are steps 6, 7, 8 or 12, 13, 14)

N_2 x {transmit and local TM process}

Where: N_1 = average number of tasks per job processed remotely,

N_2 = average number of tasks per job processed locally.

And for the case without the TM:

R_{tm}[without TM] =

{user think times} +

M_1 x {transmit and server process} +

M_2 x {transmit and local process}

Where: M_1 = average number of tasks per job processed remotely,

M_2 = average number of tasks per job processed locally.

Our conjecture is:

R_{tm}[with TM] < R_{tm}[without TM]

This is due to the TM, a machine instead of a human, doing the thinking, better load balancing and server utilization as well as feature matching. The extra messages sent by the TM for load balancing are not expected to increase the delay times noticeably. We will verify this conjecture through a series of simulation experiments.

4.4 Application Level Simulation

This section describes the TM simulation in the XNS (also referred to as NS) environment. The underlaying XNS network is a well defined component represented by parameters obtained from the XNS simulation (section 4.5). Our hypothesis (*a system with the distributed TM implemented in the application layer improves average response times compared with the current system*) will be tested. Both cases, with and without the TM, will be contrasted. It has been qualitatively argued that the proposed solution will improve the response time, however our intent is to provide quantitative results. There are several features that lend themselves nicely to simulation but on the other hand, simulating issues like transparency are difficult at best.

The simulation results are plotted to provide a graphic presentation. In order to simulate real life jobs, data about NS print servers, file servers and archiving have been collected. Workstation parameters reflect the capabilities of the Xerox 6085. Since the simulation includes anticipated future capabilities like speech and image processing, additional assumptions have been made regarding file sizes.

The simulation is done in RESQ2 V02.1986.09.22 as described in [Sau86] and available on VM/CMS. The simulation **objective** was to compare the TM with the NS system based on the parameters and variables specified earlier. Once the basic NS and TM simulators have been completed (with default parameters specified in 4.4.2), the simulation **process** was to modify the simulator to reflect the specific test case under study and simulate it with the needed parameters by using a pre-defined data file (RQ2RPLY).

4.4.1 Simulation Description

This section describes the simulation assumptions and components common to all of the simulation runs. In assessing server and workstation processing speeds the following options have been evaluated: (1) Pick hypothetical numbers. This approach has been rejected since it would not represent real life XNS servers. (2) Refer to formal product specifications to get the needed figures. The problem with this approach has been that most specifications provided lower level specifications (i.e. machine cycles per instruction, memory size...) not processing speeds as needed for the application level simulation. This is true with the exception of the printer in section 4.4.1(g2) in which its processing speed was available. (3) The last approach, which we followed in most cases, was to measure processing speeds by using available tools and user application packages. The disadvantage of this method is the lack of accuracy but it provided actual processing speed figures as close as possible to 'real life'.

(a). **The communication network.** The platform upon which the TM is simulated is XNS as described in section 4.5. Packet size remains 576 bytes thus the XNS throughput at 75 KBytes/sec will remain insensitive to the number of workstations. 75KB/sec includes retransmission and error recovery procedures. The effective throughput remains constant even at network utilization of 90% (note that based on statistics gathered for this work in XNS networks 0-116, 0-117, 0-118, (close to 300 drop cables per network, 70% of which are personal workstations) the network utilization is between 2% and 4%). The file lifetime variances are low thus the users can expect uniform response times. Therefore, XNS is represented as a single server FCFS queue processing 75KBytes per second with exponential distribution. To understand the effect of various network speeds, though, the 75KB/sec was multiplied by x2, x5, x10 and x100.

(b). **Queue types.** All of the resources implement FCFS with single server.

(c). **Error handling.** Since the XNS model accounts for retransmissions and error recovery, we will assume error free transmission.

(d). **Users.**

- It is assumed that each user has his/her own workstation.
- Without the TM, users need to perform the steps described in chapter 1 (plan processing steps, select servers...) therefore they will spend time thinking which is represented by the *think* parameter in the simulation (see 4.3). It is assumed that they will need *at least* 3 seconds but not more than 30 sec leading to the following test values: 3, 6, 15, and 30 seconds, exponentially distributed. Subsequent think times are represented by the *thinkz* parameter as defined in 4.3.

- With the TM it is assumed that 'thinking' is done by the TM. However, the users will still need 1 second (exponentially distributed) to type a 'send' command or move an icon on the screen.

(e). **Inter arrival times (IAT).** IATs (service requests) are exponentially distributed with the same exponential parameter throughout the simulation period despite the fact that it has been observed that arrivals, although random, may slightly change rates throughout the day. Worst case IAT, when all users simultaneously access the network, is defined as 100seconds.

(f). **TM site and processing needs.**

- We concentrated on simulating the distributed TM: a TM per workstation as described in chapter 3, case 1. However, the centralized TM will also be simulated. We will plot its response time versus IAT for the default values described in the next section with the purpose of understanding whether the centralized case is substantially better or worse than the distributed case. The TM implementation is done at the Application level rather than any of the lower levels (Transport).
- Adding the TM functionality will (a) require additional processing capacity from the resource on which the TM is running, or (b) the TM will contend for the existing capacity. Assuming case (b), the effect of various TM processing needs on the response time should be understood. A very conservative estimate is made to arrive at the processing time needed for the TM: the workstation message

processing time, which is discussed in (h) below, is 5 seconds for a 100KByte message. Assuming that the TM will process 30KBytes yields 1.5 seconds of TM processing time (which is 30% of the message processing time). Since the confidence level in this assumption is relatively low, it has been varied between 10% and 40%. For comparison, note that the user best case think time has been set to 3 seconds. For the centralized TM, its processing power reflect the sum of all 32 workstations: 1.5 second divided by 32 ~0.05seconds processing time.

(g). **Server characterization.** We made the assumption that servers are always available and proceeded to evaluate their actual processing speeds.

(g1). The available file servers are made of the aging Century 300 disks and the proprietary-hardware called DLion running filing software as described in [Xer03]. Processing figures are non-existent therefore the only choice left was to measure it. It has been difficult to accurately measure since this figure is tightly coupled with the transmitting station and the communication network's throughput. Nevertheless, using the SPY tool (see below) we ascertained that only one workstation was active on the network. The workstation was fully dedicated to transmitting various size files to the file server. We observed the transmission of over 20 MByte of data in over 70 files, yielding file server average speed of 52KBytes/sec. (The SPY tool is a utility running on a workstation and monitoring network activity. For each transaction, it displays the originating and destination addresses and indicates when collisions occur. File sizes were known thus we measured the transmit/receive time for short (2KBytes) and long (280KBytes) files).

(g2). For print servers the Xerox 4050 machine was considered. Its processing speed is 50 pages a minute for a page containing text and images. 30KB is requires to represent a complex page therefore the processing speed used is 25KB/sec (50pages a minute x 30Kbytes a page/60). The difficulty in rating any graphic imager in absolute numbers is acknowledged.

(g3). We proceeded to characterize the compute servers which are VAXs: 8200, 8350, and a cluster of two 8800s running VAX/VMS. Hence processing speed assumptions are made based on the available VAXs. Other servers may also be available thus specifying processing speeds may be a matter of choice. Therefore, we will assume processing speeds of 200KBytes/sec based on using the 8800 cluster. This figure was obtained by running a conversion program that converted raster images from 75 dots per inch to 300 dpi.

The simulation includes 5 servers which have been divided into two groups: {ser_1, ser_2, and ser_3} and {ser_4 and ser_5}. Ser_4 and ser_5 are designated as the compute servers. Their processing power differ from each other in order to simulate an **asymmetrical system**. Ser_1, ser_2, and ser_3 represent filing, filing/archiving, scanning and printing. Several printers are available to support the user population in the networks mentioned earlier hence we could require that all of these servers will represent printers. However, we will assume that no more than one server is a printer and the rest are filing, archiving, or scanning servers. The servers are insensitive to the job source. In summary, the server processing capacities are: $ser_2 = 2x10^{-5}$ sec/KBytes (based on 50KB/sec as explained earlier); $ser_3 = 4x10^{-5}$ sec/KBytes (based on 25KB/sec as explained earlier); $ser_5 = 5x10^{-6}$ sec/KBytes (based on 200KB/sec as explained earlier); $ser_1 = 1x10^{-5}$ sec/KBytes (twice as fast as ser_2); $ser_4 = 1x10^{-6}$ sec/KBytes (5 times faster than ser_5 to provide for an asymmetrical system). Note that these figures were varied to study the effect of server processing speeds on the response time.

(h). **Workstations.**

The workstations are Xerox 6085s: proprietary 32 bit Mesa processor, 80MBytes hard disk 30% full, 2.2MB RAM, running ViewPoint (VP) and Xerox Development Environments which are the user environments. Workstation processing speed has to be determined. Quoting MIPS or MFLOPS are useless and would not suffice, therefore real life experiments with available utilities have been performed. Several files have been paginated, converted from VP format to ASCII and then from ASCII to VP (see table below). Note that not all of the document features could be converted e.g. vectors, font information, etc.

Therefore following the first conversion, the files have been converted back and forth twice and average times have been recorded. Also note that the original file sizes changed by a factor of up to 3, i.e. <file 2> changed from 40KBytes to 16KBytes, <file 10>, from 2KBytes to 1.5KBytes. With the exception of file size changes, the results have been tabulated below. Pagination rate is in seconds per KByte.

FILE	SIZE	PAGINATE		CONVERT TIME	
NAME	[KB]	TIME	RATE	VP->ASCII	ASCII->VP
<FILE1>	40KB	30SEC	.75	30SEC	20SEC
<FILE2>	40KB	30SEC	.75	30SEC	23SEC
<FILE3>	120KB	65SEC	.55	41SEC	40SEC
<FILE4>	104KB	75SEC	.72	39SEC	34SEC
<FILE5>	57KB	21SEC	.37		
<FILE6>	32KB	35SEC	1.1		
<FILE7>	34KB	30SEC	.88		
<FILE8>	49KB	45SEC	.92	35SEC	29SEC
<FILE9>	77KB	40SEC	.52		
<FILE10>	2KB	3SEC	1.5	15SEC	15SEC

With the exception of <file10> the conversion times are smaller or equal to the pagination times. The pagination rate ranges from .37sec/KBytes upto 1.5sec/KBytes with the average around 0.8sec/KBytes. The average convert time of the first 4 files is 0.4sec/KBytes (risking the inaccuracy of calculating conversion rates based on the original file size, not the actual one as discussed above).

The spelling check rate test, not included in the table, yielded 0.5sec/KBytes. Another set of measurements indicated that disk access and driving software to prepare for transmission takes 2 seconds (measured manually) regardless of file size. This is the amount of time from initiating the filing request at the keyboard till first packet appears on the Ethernet.

The workstations generate message and control packets to be transmitted over the Ethernet. The XNS control packet length is ~500 bytes. The message length (job size) is 100KBytes as stated later. Therefore the simulated workstation includes two classes with exponential distribution parameters: one for control packets and one for data packets.

The average processing time for control packets is calculates as follows: All of the measurements taken include heavy disk access which is slower than CPU or memory speeds. Disk access will not always be needed for control messages. Therefore $t'_{control} = [(0.8 + 0.4 + 0.5)/3]x0.5 = 0.28sec$ may be an order of magnitude bigger than reality thus we will use $t_{control} = 0.05sec$. Note that $(0.8 + 0.4 + 0.5)/3$ is the average of all three figures discussed earlier. The message processing time is composed of: 2 seconds for disk access time as described earlier plus $[(0.8 + 0.4 + 0.5)/3]x5 = 2.8sec$ assuming 5KB worth of processing needed prior to file transmission. Thus $t_{message} = 2 + 2.8 = 4.8sec \rightarrow 5sec$.

The response time sensitivity to workstation processing power variances has been studied. This is especially important in case the TM is allowed to assign tasks to the originating workstations, referred to as cooperative workstations. This parameter was varied assuming that workstation processing power may only increase. The number of workstations is varied from 8 to 32.

A feature in RESQ, called Job Vectors (JVs), is associated with each simulated job arrival and may be treated by the simulation programmer as needed. The workstation module includes a SET node which assigns source numbers to the jobs generated at the particular source by using JV(1). RESQ initials all JV's to zero at the beginning of every run. The job sources are S_1 for WS_1, S_2 for WS_2, . . . S_{32} for WS_{32} with variable arrival rates.

(i) **Load balancing.** Load balancing techniques have been described in chapter 3 and include the following:

- Random (static),
- Server capacity threshold (dynamic),
- Server capacity relative (static),
- Server overflow (dynamic), and
- Server overflow, optimized (dynamic).

(j). **Job size and its processing time.** The next parameter which needed attention was job size. The average file size in the table above is 55KB. Knowing that incorporating scanners, voice and video digitizers will imply larger files, we have selected 100KB per file. This is done based on knowing that one 8.5x11 inch page scanned at 300 dots per inch and compressed 10:1 will result in 100KB and that telephone voice digitizers convert at 600KBytes per minute which yields 100KB a minute compressed only 6:1. This parameter is changed for sensitivity checks as described in section 4.4.2. For constant job sizes, their processing times may vary thus this parameter is also changed as described in 4.4.2.

(k). **Simulator components.** The simulator is composed of the following components and modules:

- Up to 32 user and workstation modules.

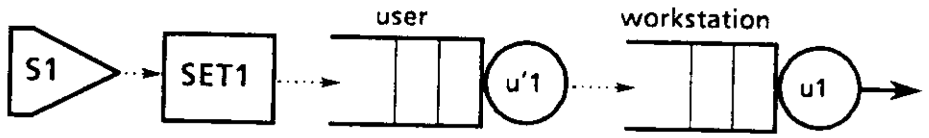


Figure 4.3.1a: User and workstation module

Arrivals, as described earlier, are represented by Source nodes. Each workstation receives jobs from its associated source as depicted in figure 4.3.1a. The Set nodes assign workstation ID to each workstation. User actions are represented by one FCFS queue called *Human* which includes two classes: (1) initial user think time and (2) subsequent think times as described earlier. The workstation is represented by the *workstation* queue with two classes: one for *status* processing and one for data files processing, each with its processing times described earlier.

- The network.

  The network is represented by the Net queue (see figure 4.3.1) with classes for status and data files. Several status and data file classes are needed for routing purposes all sharing the same processing rates: status processing rate is 1/150 second per status file (0.5KB at 75KB/sec), and 4/3 seconds for data files (100KB at 75KB/sec).

- And the five servers discussed earlier. The SETX node is needed to distinguish packets which have been processed at least once from newly arriving ones.

(I). **Job routing.** Job routing is accurately described in the CHAIN section of the RESQ code in the appendix. However, to demonstrate it, a brief description for the NS case with one active workstation follows (figure 4.3.1): ***S1*** to ***SET1*** (which assigns WS number to each job) to ***USER1*** (thinking) to ***WS1*** (STATUS-CLASS, prepare and send a status message) to ***Net*** (STATUS CLASS, network transmission time for status packets) to ***Net*** (STATUS CLASS, network transmission time for status packets) to ***dum3*** to ***USER1*** (thinking) to ***WS-MESSAGE-CLASS*** (workstation partial message processing) to ***Net*** (MESSAGE CLASS, network transmission time for messages). An assumption that servers response times to the status query is negligible is made. At this point 'new' jobs are routed to the

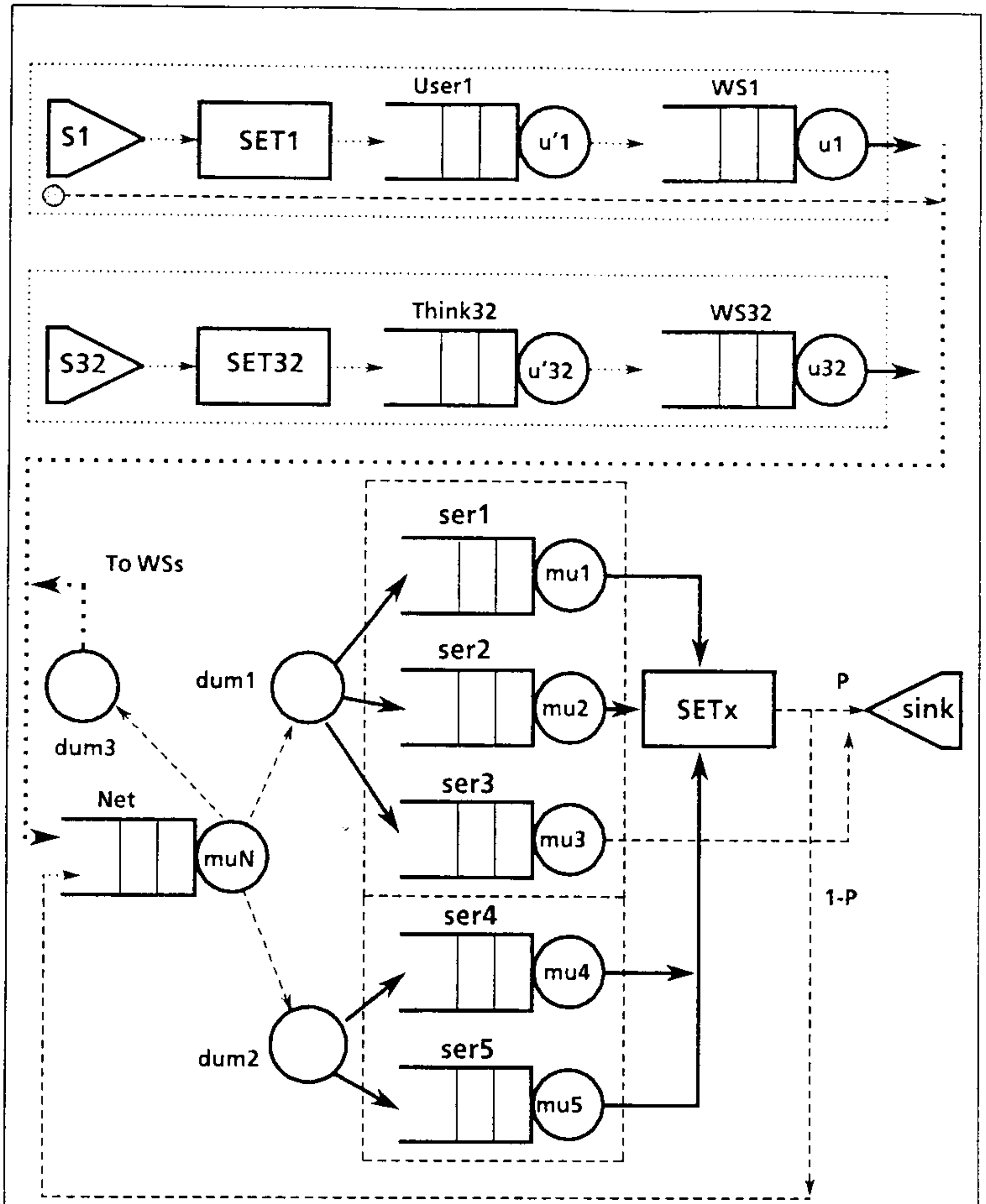


Figure 4.3.1: Basic RESQ modules

compute servers with probability 1 and each server receives 50% of the jobs. 'Old' jobs are assigned to *dum1* with probability 0.4 and to *dum2* with probability of 0.6. Servers 1, 2, and 3 receive 1/3 of the jobs each. Jobs leaving *SER3* are terminated. All other jobs are routed to ***setx*** (setx marks jobs as 'old' for routing) and terminate with probability of 0.7. The rest, 30%, are routed back to the network and the user for further processing and repeat the same routing. A final note: with the TM, routing was essentially similar with a major exception: load balancing affected routing to the resources.

(m). For simplicity, the optional Done messages are turned off.

(n). Third party data transfer is not available.

The result of varying the simulation parameters will affect the following:

(1) Server utilization.

The effect of the above on server utilization should be understood.

(2) Bottlenecks.

The introduction of the TM and its management procedures should not create bottlenecks which severely affect response times thus any potential bottleneck should be identified. Further studies should ascertain and quantify the improvement in response time due to the removal of a bottleneck.

(3) Communication overhead.

The TM generates extra status packets for load balancing. These packets are considered overhead traffic and the resultant effect on the response time should be understood. This can be performed by analyzing the communication network delays obtained from the simulator.

(4) Relative cost.

Although it has not been our objective, the simulation results will enable a cost tradeoffs discussion.

4.4.2 Results, Analysis and Conclusions

The following sections show the simulation results contrasting the TM with the current implementation referred to as XNS (or NS). The simulation process, described earlier, was as follows: A model was run using *EVAL* and *RQ2RPLY* files. Then the parameters under study were changed to produce the next simulation. Regarding notation: R_{tm}[TM] means response time for the TM case. If additional information is needed, it will also be included in the brackets.

All simulations use the following default values unless specified otherwise: (for further description refer to section 4.3):

PARAMETER	NS	TM
Number of workstations	32	32
Network speed	75KB/S	75KB/S
Load balancing	random	dynamic, server capacity threshold
Job size	100KB	100KB
think time	3sec	1sec
thinkz time	3sec	1msec
workstation message processing time @100KB/file	5sec	5sec
Status processing time (workstation)	0.05sec	0.05sec
TM processing time with any dynamic load balancing algorithm	n/a	1.5sec
TM processing time with any static load balancing algorithm	n/a	1.0sec

The following sections describe each simulation's **objectives**, **procedure**, **results**, **analysis** and **conclusions** relative to the particular simulation.

4.4.2.1 Concept viability

Objective: Obtain R_{tm} for NS and TM (with default values) to demonstrate whether the TM concept is viable at all in the given environment described in 4.4.1.

Procedure: (1) Simulate NS for 32, 24 and 8 users with the default parameters.

(2) Repeat step (1) with the distributed TM implementing dynamic capacity threshold load balancing, described in 3.3.

Results: Figure 4.3.2 shows R_{tm} as a function of inter arrival times (IAT) for 32, 24 and 8 users for NS. With the exception of 8 users, and maintaining IAT<100 seconds, R_{tm} drops sharply as IAT increases. For 24 users, R_{tm} = 230sec @IAT = 40sec dropping to R_{tm} = 25sec @IAT = 100sec. The drop is similar for 32 users: R_{tm} = 182sec @IAT = 50sec and 32sec @IAT = 100sec. For IATs between 100sec and 1000sec R_{tm} drops from the values mentioned earlier much more moderately. For IAT = 1000 sec and higher R_{tm} stabilizes at 19 sec. In the case of 8 users, R_{tm} drops very slowly (<5%) as IAT increases from 70sec to 1000sec.

Figure 4.3.3 shows R_{tm} as a function of IAT for 32, 24 and 8 users with the TM introduced. As with the NS, the curves show a sharp drop in R_{tm} for 32 and 24 users in the range of IAT = 50sec to 80sec after which R_{tm}[24users] stabilizes at 14seconds. R_{tm}[32users] stabilizes at the same value only at IAT = 120sec. It is also observed that for 8 users R_{tm} is virtually constant at 14sec for IAT = 70 seconds and higher.

Figures 4.3.4, 4.3.4a, 4.3.5 and 4.3.6 contrast the NS with the TM curves and are analyzed in the following section. Figure 4.3.7 includes several plots. R_{tm} is given for reference. The network utilization is over 99% for IAT $\leqq$ 70sec, drops to 80% @IAT = 100sec and further drops to ~40% @IAT = 200sec. The network and system throughputs are depicted as well. The network throughput is higher than the total system throughput.

The simulation also provides server utilization rates and network mean queuing times. The server utilization rates are very low at all IATs for both cases - TM and NS. For example, NS figures are: average of all five servers is less than 5% at the high IAT of IAT = 60sec with maximum of 14.6% (server 5) and minimum of 2.3% (server2). At IAT = 1000sec the average is 0.4%(!). Mean network queuing time is 0.7sec at IAT = 1000sec, 2sec @IAT = 120sec, 5sec @IAT = 90sec. Mean network queuing time indicates the amount of time a job spends in the network. Note that a job may be transmitted over the network more than once. For NS and the TM, the workstation utilization is very low at all IATs: highest is 0.26 at IAT = 80 sec in the entire simulation for this test.

Analysis: Figure 4.3.4 contrasts the NS case with the TM for 32 users. We observe that for $IAT<60sec$, adding the TM substantially increases the response time. The reason for this increase, as supported by figure 4.3.7, is the very high (0.99) network utilization which introduces a bottleneck. Even a minimal increase in the network traffic due to the needed status messages for load balancing causes a large increment in response time. The load balancing algorithm does not compensate for the added network delays. The NS versus TM break even point is at IAT = 60 seconds above which the TM's load balancing feature significantly reduces R_{tm} compared with NS. At IAT = 80sec the TM response time is 60% of the NS time: 30sec compared with 50sec - a difference of 20 seconds. At IAT = 150sec the TM's response time is lower than the NS by 6 seconds yielding 75% of R_{tm}[NS], and for high IATs, 500 seconds and higher, the TM still improves performance: 14sec versus 19sec, a difference of 5 seconds yielding 74% of R_{tm}[NS]. This is depicted in figure 4.3.4(a). Note that for $IAT<60sec$ the curve in figure 4.3.4a would yield negative numbers since the TM increases the response time. Again, we notice that the load balancing algorithm is effective the most when there is a large number of arrivals ($65sec<IAT<85sec$) facilitating a substantial delay reduction, higher than 15 seconds.

The same analysis is applicable to figure 4.3.5 which contrasts the case with 24 users. Both curves drop sharply as IAT increases from 60 seconds. We observe that the break even point (NS versus TM) is between 45sec and 55sec inter arrival times. The break even point for R_{tm}[32] is at IAT = 60 seconds. The reason for the shift is: for a given network utilization, a reduced number of users will need to generate a higher number of arrivals.

Figure 4.3.6 is different than the previous two figures since it shows that the TM improves R_{tm} for all IATs tested. The break even point (TM versus NS) for 32 users is at IAT = 60 seconds, for 24 users - at IAT~50sec, thus it is assumed that the break even point for 8 users is at IAT<40 seconds (by calculation at IAT = 15sec). Reducing the number of users for a given IAT means reducing the network traffic thus preventing a bottleneck. We also observe that regardless of the number of users, R_{tm} is reduced from 19 to 14 seconds due to the introduction of the TM. The curves are flat since (a) the network doesn't create a bottleneck and (b) all the servers operate at low utilization as indicated before. Note that as expected and regardless of the number of the workstations, R_{tm}[TM] for high IATs converges at the same final figure. The same observation is correct for R_{tm}[NS].

The system throughput (TP) curve shows throughput dropping as IAT increases. Knowing that each arrival created a departure -no jobs are being discarded- this curve should follow TP$\leqq$32/IAT, which it does.

An important note, though, is that in this network IATs of 100 or lower are unrealistic. Users cannot sustain this rate of job submission. Therefore the TM will be effective in real life. Although the interest is in understanding the real life situation, IATs of less than 100 seconds will be analyzed.

The NS simulation assumes perfect "random" job allocations as described in 3.3 where this may not be the case thus in reality the NS numeric results may be slightly worse.

Conclusions: (a) It is evident that the TM, although increasing network traffic, reduces the parameter that users are most sensitive to: response time. It is reduced from 19 to 14 seconds for the expected real life low arrival rates of IAT>500sec. The communication overhead does not nullify the TM's response time improvements.

(b) If IAT drops below 100 seconds, the network becomes a bottleneck. {Therefore the next simulation will address this issue.}

(c) The network servers in the simulated system are underutilized. There is a possibility to reduce system cost ($) by using less powerful servers without substantially increasing response times. However, this conclusion will be tested later as well.

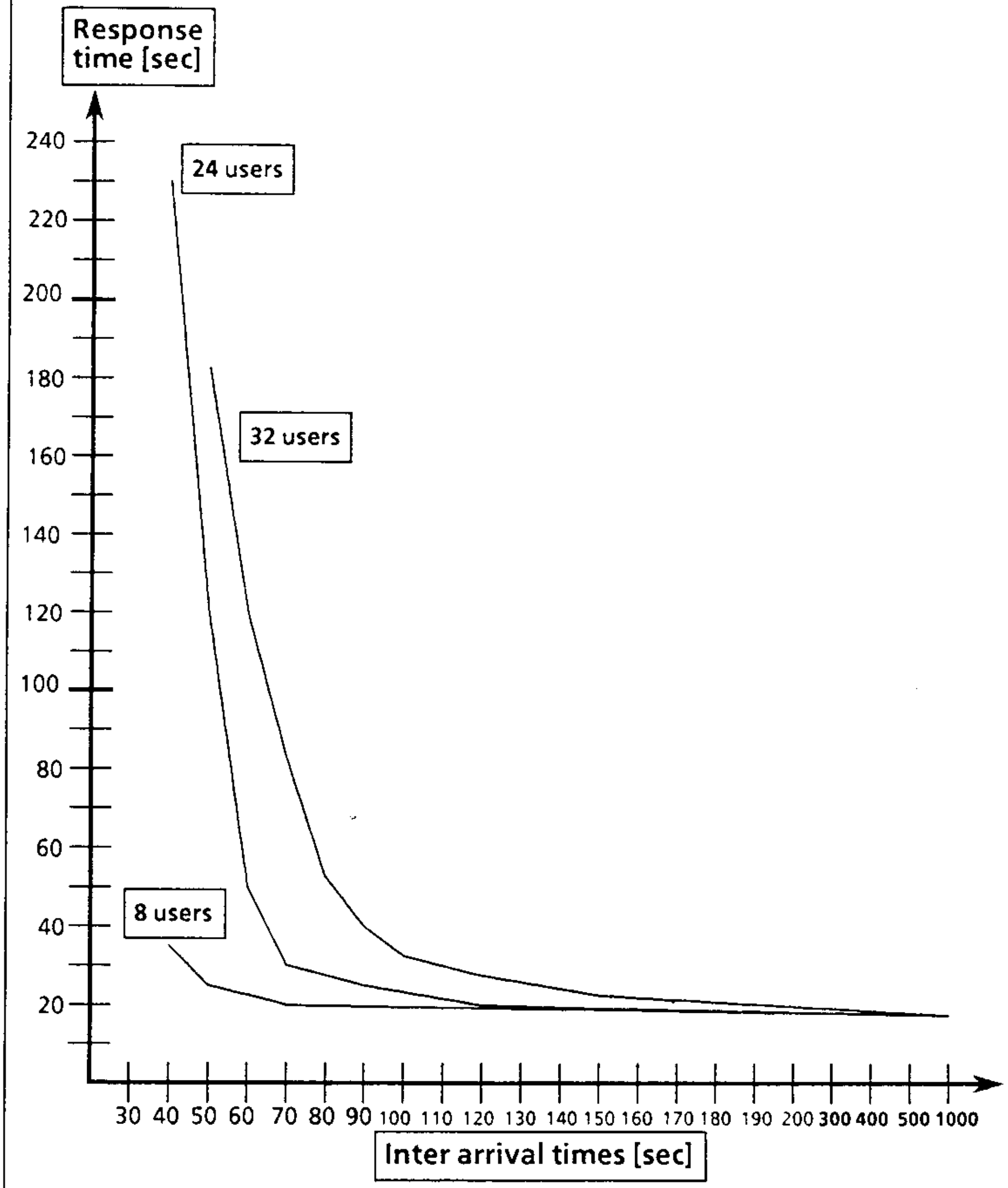


Figure 4.3.2: NS Response times

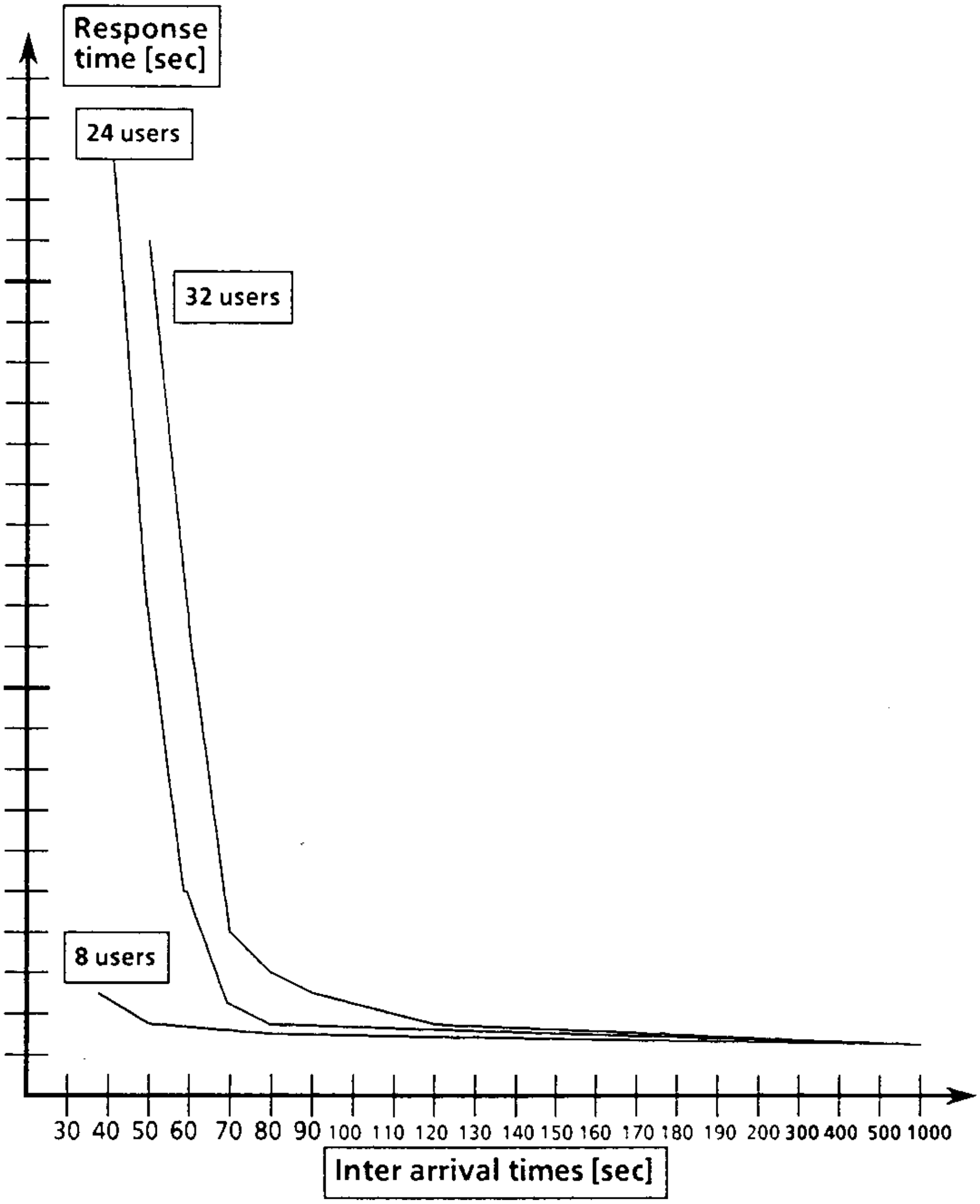


Figure 4.3.3: TM response times

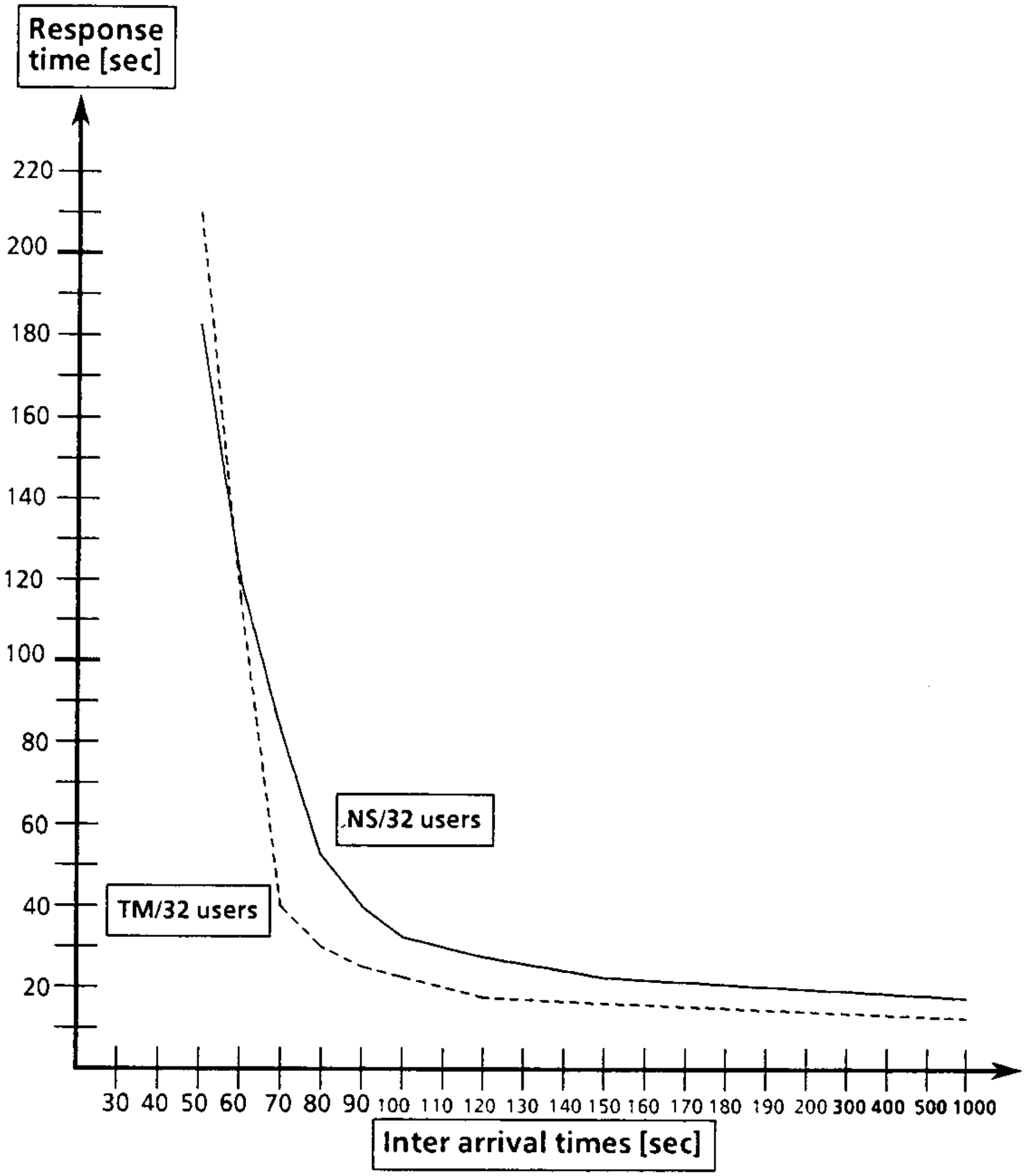


Figure 4.3.4: NS and TM response time comparison / 32 users

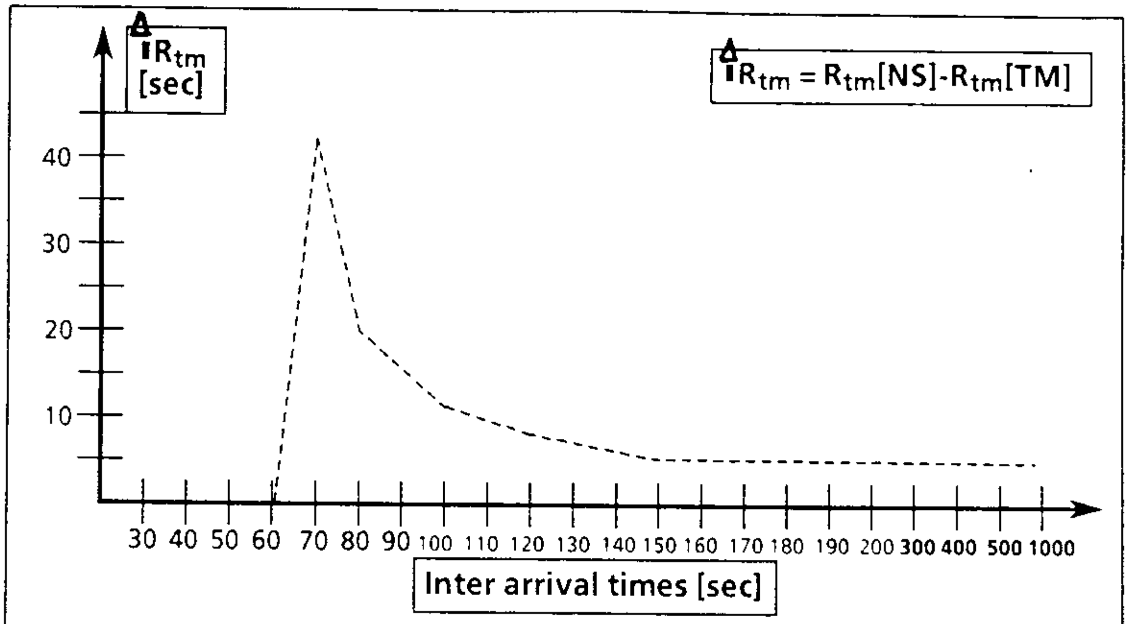


Figure 4.3.4(a): NS versus TM, R_{tm} differential / 32 users

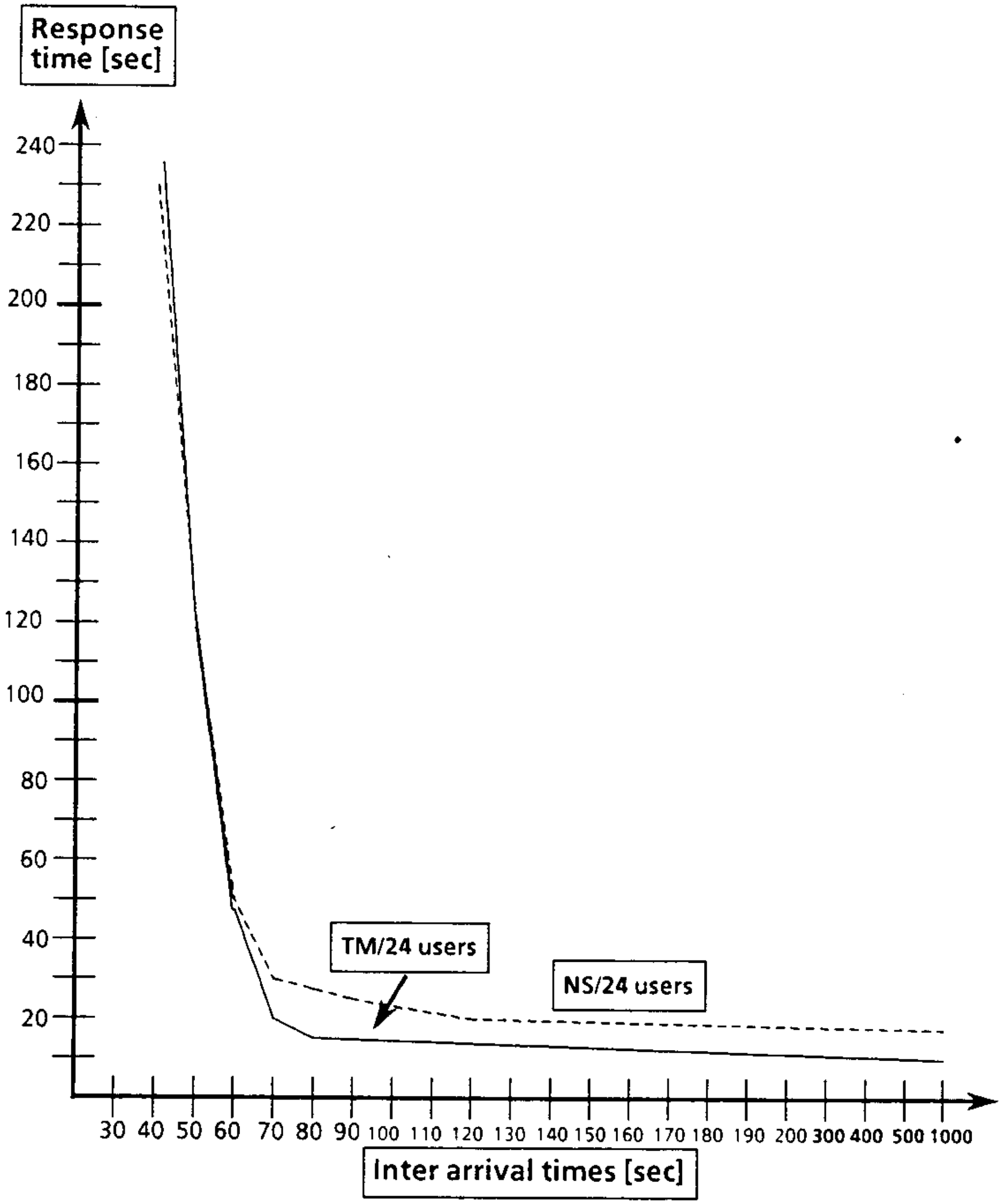


Figure 4.3.5: NS and TM response time comparison / 24 users

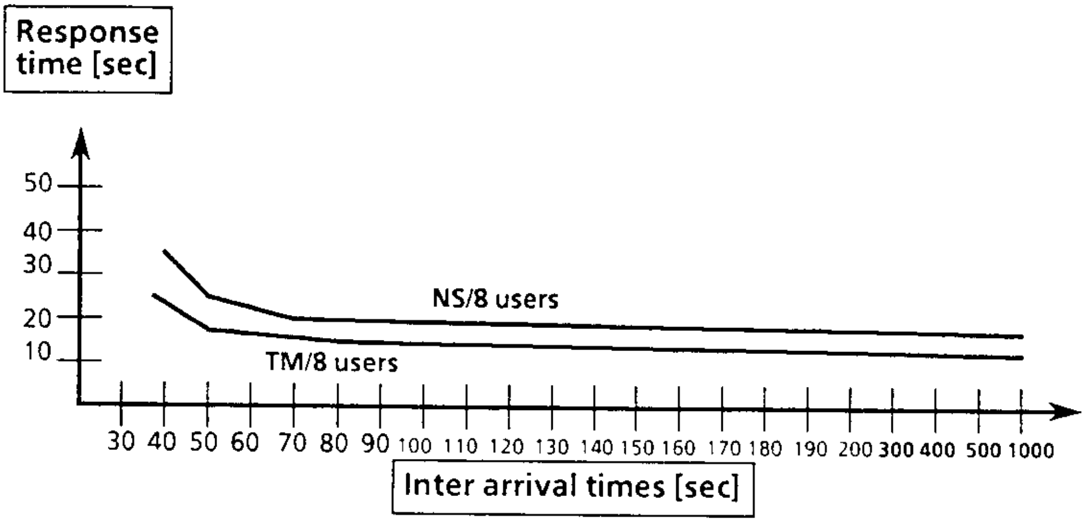


Figure 4.3.6: NS and TM response time comparison / 8 users

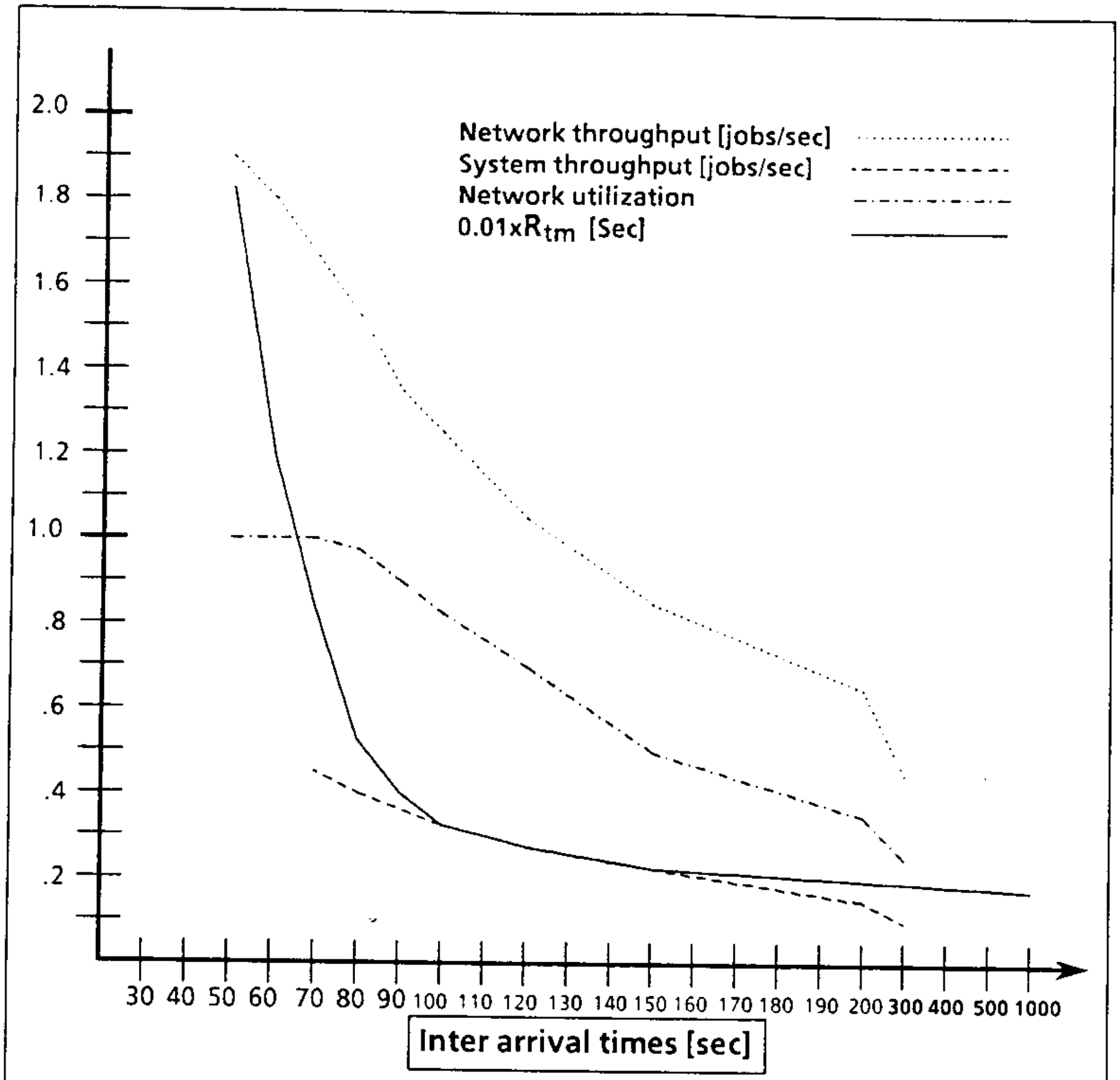


Figure 4.3.7: TM throughput, response time and utilization

4.4.2.2 Effect of network speed changes

Objective: Understand the effect of increasing network speed on R_{tm} and mean network queuing times (QT) for NS and TM.

Procedure: Simulate NS and the TM at worst case of 32 users for network speeds of up to 100 times faster than the current implementation of 75KBytes/sec.

Results: Based on the simulation model developed for 4.4.2.1, the network speed has been increased while keeping the rest of the parameters unchanged. The simulation was run for relative network speeds of x2, x5, x10 and x100 (normalized at 75KB/sec as x1). R_{tm} and QT obtained for relative network speeds of x100, x10 and x5 reveal that the results for x100 and x10 do not substantially differ from each other and furthermore, as depicted in figure 4.3.8, R_{tm}[NS] for x10 and x5 are quite similar (within ±5%) for IAT>70sec. R_{tm}[TM] follows the same pattern. Therefore to provide a meaningful discussion, the results covered here include relative network speeds of x1, x2, and x5.

Figure 4.3.8 plots R_{tm}[NS] as a function of IAT for network speeds of x1, x2, x5, and x10. We observe that for all network speeds greater than x1, R_{tm} @IAT = 1000seconds decreases from 19 to 16 seconds. For IAT>70sec the x5 curve is constant at 16sec while the x2 curve gradually drops from 22sec

@IAT = 70sec to 16sec @IAT = 1000sec. The results obtained for the TM case reveal the same trend and are plotted in figure 4.3.9. Note that this figure does not include network speeds of x5 nor x10. The x5 plot is drawn in figure 4.3.10 which is effective in comparing all four combinations of TM, NS at network speeds of x2 and x5. Table 4.1 lists the mean queuing times for the TM at relative network speeds of x1, x2, and x5. It clearly shows the large drop in queuing times even if the network speed is increased from x1 to x2. Workstation and server utilization remained low as described in 4.4.2.1.

Table 4.1: Queuing times for network speeds of x1, x2, x5.

IAT	Network Queue Time[sec]		
[sec]	x1	x2	x5
50	35	1.5	0.23
60	20	1.2	0.19
70	9	0.9	0.18
80	6	0.7	0.18
90	5	0.7	0.17
100	4	0.7	0.17
120	2	0.5	0.16
150	1.6	0.5	0.15
200	1.5	0.4	0.1
300	0.9	0.4	0.1
500	0.7	0.4	0.1
1000	0.6	0.3	0.1

Analysis: From figures 4.3.8 and 4.3.9 it is clear that regardless of whether the TM is introduced or not, doubling the network speed will greatly affect R_{tm} at IAT$\leqq$80seconds. For the NS case at IAT = 80sec, R_{tm}[NS, x1]~50seconds while R_{tm}[NS, x2] ~20seconds and R_{tm}[NS, x5]~16seconds. This trend is similar for the TM and points again to the high network utilization rates and queuing times as described in the previous table.

A major advantage of the TM is revealed in figure 4.3.10:

$$R_{tm}[TM, x2] < R_{tm}[NS, x5] \text{ for IAT} \geqq 50\text{seconds}$$

This is due to the effect of load balancing which is more pronounced as the network delays get reduced. But, again, for IAT$\leqq$50sec; R_{tm}[NS, x5]<R_{tm}[TM, x2] due to the reasons explained before.

Another observation: decreasing inter arrival times from 60 seconds to 40 seconds increases R_{tm}[TM, x2] by ~8seconds while R_{tm}[NS, x2] increases by ~17 seconds. This is again, due to the load balancing feature of the TM.

Conclusions: (a) Increasing the network speed by 100 or even 10 is not justified since increasing it by x5 will eliminate the bottleneck anyway.

(b) Considering response time, it is better to increase the network speed x2 and add the TM than increase the network speed x5 without adding the TM. It may also be more economical. Said differently, introducing the TM will allow the network speed to be doubled rather than multiplied by 5 and yet achieve better response time. Increasing XNS speed by 2 is feasible (changing packet size to 1500bytes, see section 4.5).

(c) Mean network queuing time is greatly reduced merely by doubling the network transmission speed.

(d) At network speed of x1 and low IAT (<70sec) the network becomes a bottleneck, as concluded in 4.4.2.1.

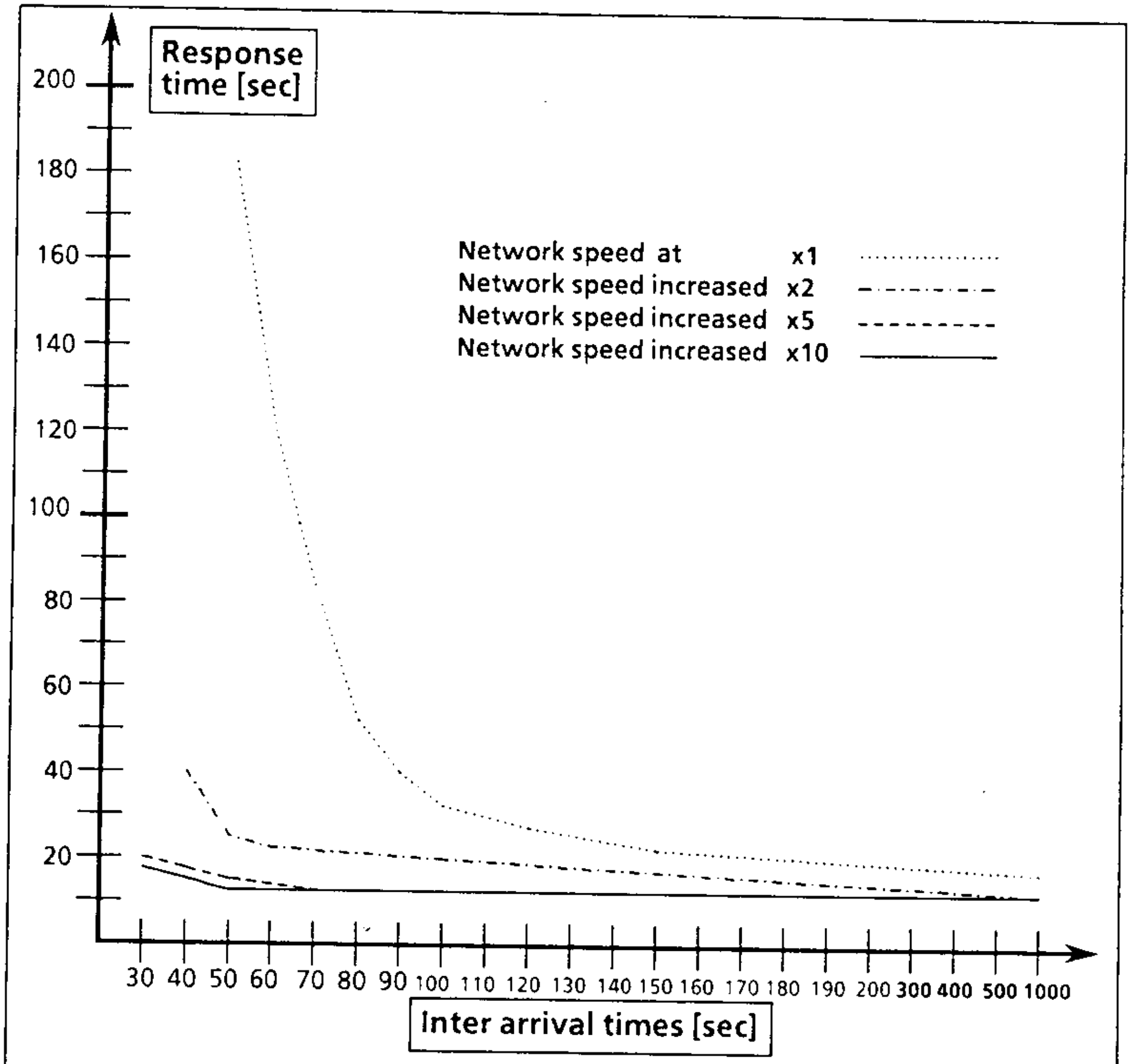


Figure 4.3.8: Effect of network speed changes on response time , NS

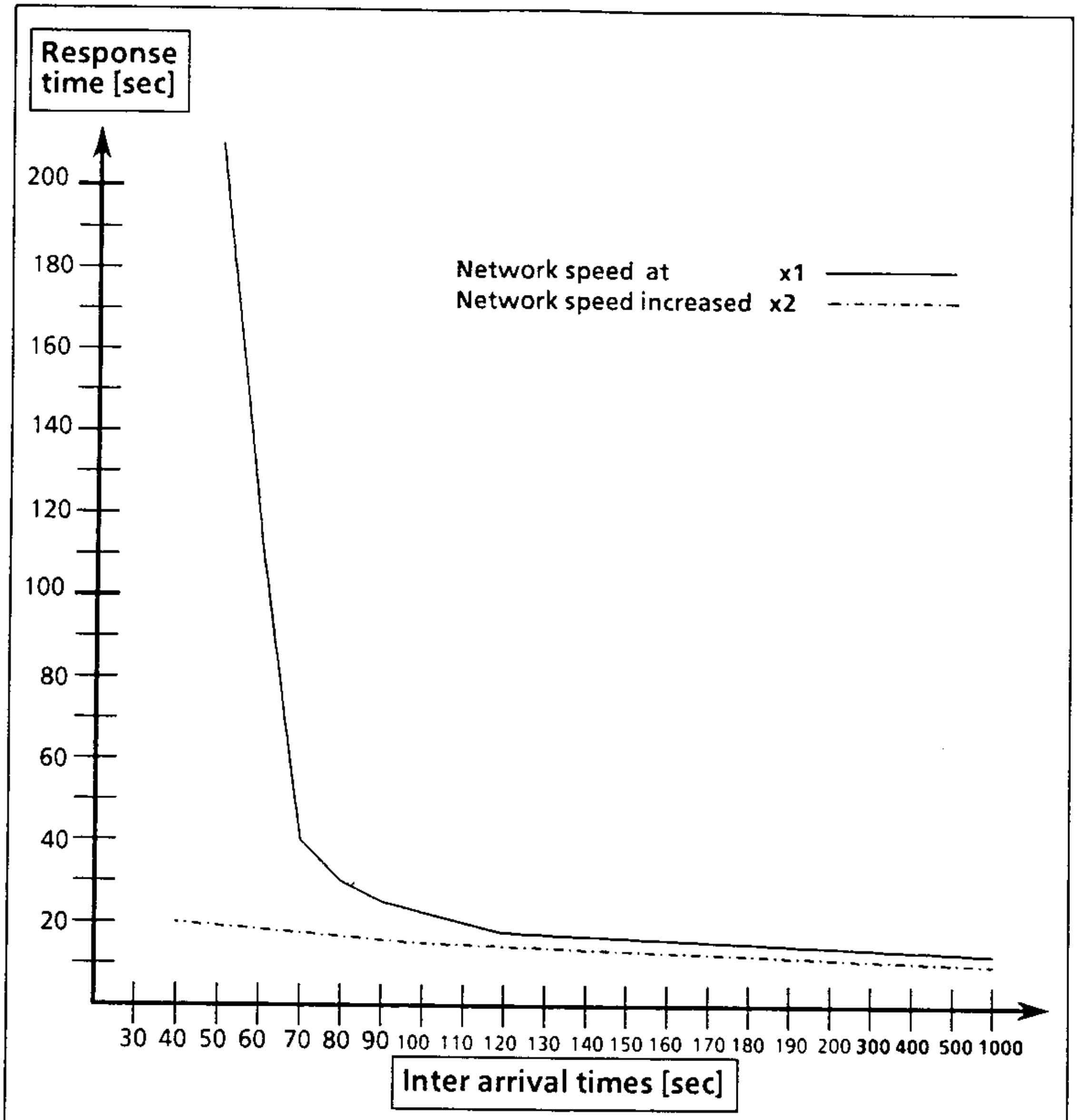


Figure 4.3.9: Effect of network speed changes on response time, TM

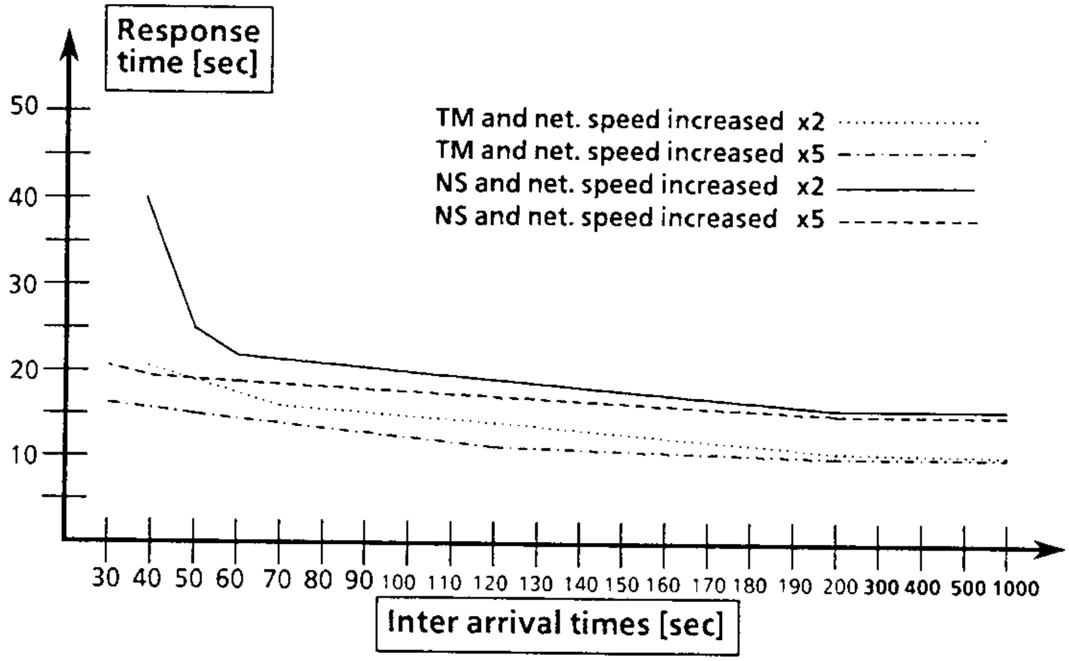


Figure 4.3.10: Effect of net. speed changes on response time, NS vs TM

4.4.2.3 Effect of load balancing algorithms

Objective: Define R_{tm} improvements as a function of the load balancing algorithms under investigation.

Procedure: Simulate the algorithms described in 3.3 to obtain R_{tm} versus IAT curves for the default values defined in section 4.4.1.

Results (1): Figures 4.3.11, 4.3.12 and 4.3.13 plot R_{tm} as a function of IAT for the given algorithms. Due to the network being a bottleneck at high arrival rates of IAT<100seconds, several simulations have been performed at a network speed of 10 times the default value of 75KB/sec.

Figure 4.3.11 shows the "no load balancing" curve. It is assumed that NS users will randomly submit jobs to servers without any network entity directing their submission. Therefore, each server, regardless of its processing capacity, will be assigned the same number of arrivals. The dynamic server capacity threshold and the static server capacity relative algorithms are also plotted in order to provide a comparison. Note that the static algorithm requires less processing time since it does not require *status* message processing. For IAT$\geqq$120 seconds, the static algorithm provided better response times. For 60<IAT<120 seconds the dynamic capacity threshold algorithm provided the best results but for IAT<60 seconds, the static algorithm was slightly better. Server utilization has been discussed in 4.4.2.1

for random and server capacity threshold algorithms. Server utilization for the capacity relative algorithm has been identical to the dynamic algorithm. Mean queue lengths for all servers (network excepted) were between 0.15 and 0.7.

Analysis(1):

- The **static** algorithm performs better @IAT>120 seconds because:

  (1) The TM requires less processing time: there are no status messages to send, receive and process,

  (2) There aren't enough jobs to perform load balancing.

- The **static** algorithm performs better @IAT<60 due to the network bottleneck severely impacting the dynamic load balancing algorithm which relies on the status messages to perform.
- The **dynamic** algorithm performs better at 60<IAT<120 seconds since the network does not create a bottleneck, and there are more arrivals than for IAT>120 seconds.

Conclusions(1): 'No load balancing' provides the worst R_{tm} at all IAT ranges under study. The capacity relative algorithm is superior in the (real life) range of IAT>100seconds.

Results(2): Figure 4.3.12 compares the static capacity relative, dynamic capacity threshold (t/h) and the dynamic optimized queue overflow algorithms. The dynamic optimized queue overflow algorithm is tested with 2 maximum queue lengths: 1 and 3. We observe that this algorithm performed equally well regardless of its max. queue length. For the range of IAT roughly between 60 and 100 seconds, the capacity t/h algorithm performed the best. For IAT<60 seconds, both dynamic algorithms performed the same.

Analysis(2):

- For IAT>120 sec the static algorithm performed the best as discusses above.
- For 60<IAT<120sec the differences between all dynamic algorithms were too little to judge but the simulation results slightly favored the capacity t/h. Nevertheless, the servers are not busy to a degree facilitating fine comparison. Note that extreme queue lengths varied between 0 and 3 with average (at IAT = 80 seconds) of 0.5.
- For IAT<60 sec the static algorithm performed the best as discusses above.

Conclusions(2): 'No load balancing' provides the worst R_{tm} at all IAT ranges under study.

When the communication network may create a bottleneck, or for low arrival rates (IAT>120sec) the static algorithm is preferred. Since it's assumed that real life IAT will be greater than 100sec, the static algorithm should be selected.

Results(3): Due to the two results discussed above the network speed was increased by a factor of 10. The results are plotted in figure 4.3.13. Note that the simulation accuracy did not allow to accurately draw the static and dynamic curves. See later note on simulation results. As previously shown, even with increased network speeds, random load balancing ('no load balancing') performed the worst.

Analysis(3): This analysis is divided into two parts based on whether IAT is lower or higher than 130 seconds. For IAT>130 sec, the reasons discussed above for IAT>120sec apply. For IAT<130 sec, the dynamic load balancing algorithm is slightly better (in terms of R_{tm}) since the network does not create a bottleneck thus facilitating the passage of status messages needed by the dynamic algorithm.

Conclusion(3): 'No load balancing' provides the worst R_{tm} at all IAT ranges under study.

With the network not being a bottleneck, the dynamic capacity t/h load balancing algorithm should be selected for IAT<130sec.

Results(4): The server overflow load balancing algorithm was tested. Its results were worse than the random algorithm therefore it is not included in this work.

Additional conclusions: These experiments help analyze the effect of the extra communication overhead generated by the dynamic load balancing algorithms. Aside from the algorithm itself, a difference between simulating any dynamic and static algorithms (with the exception of no TM) is: with the static algorithm no status messages are sent thus the TM execution time is reduced by 0.5 second. The status message transmission time is 0.5% of the data message (1/150 versus 0.75). The extra status messages should affect the response time when the network is utilized the most, at IAT<60 seconds. However, analyzing the plots (figure 4.3.11) reveals that the difference between the static and dynamic R_{tm}s is unnoticeable. This difference is due to both the communication overhead and the difference in TM execution times, therefore, we conclude that the communication overhead is negligible.

The conclusion is also supported by the following calculation:

Problem formulation: calculate communication overhead in terms of delay time (T) due to the transmission of the extra status messages needed by dynamic algorithms.

Given:

- Network is m/m/1 with 4/3 sec average processing time per data message and 1/150 sec for status.
- Each data message will terminate with probability 0.7 and sent back to the servers with probability 0.3.
- Each data message requires average of two status messages.
- 32 workstations generating jobs randomly (Poisson arrivals).
- T(m/m/1) = $1/\mu-\lambda$ [Kle76]

  where μ is the server processing power and λ the arrival rate.

- Overhead is defined as the time difference between T_2(time delay with status messages) and T_1(time delay, no status messages).

Solution: Average number of times a message is transmitted over the network is $k = \Sigma\,(2n-1) \times 0.7 \times 0.3^{n-1}$ for all n. Thus $k \sim 1.8$.

$T_1 = 1/(0.75 - 32 \times 1.8 \times \lambda)$ thus $T_1 \sim 1/(0.75 - 57 \times \lambda)$.

With overhead: $1/\mu = 4/3 + 2 \times (1/150)$

$\mu = 150/202$

$T_2 = 1/\{(150/202) - 57 \times \lambda\}$

Thus: T_2-T_1 (@IAT = 100) < 1second.

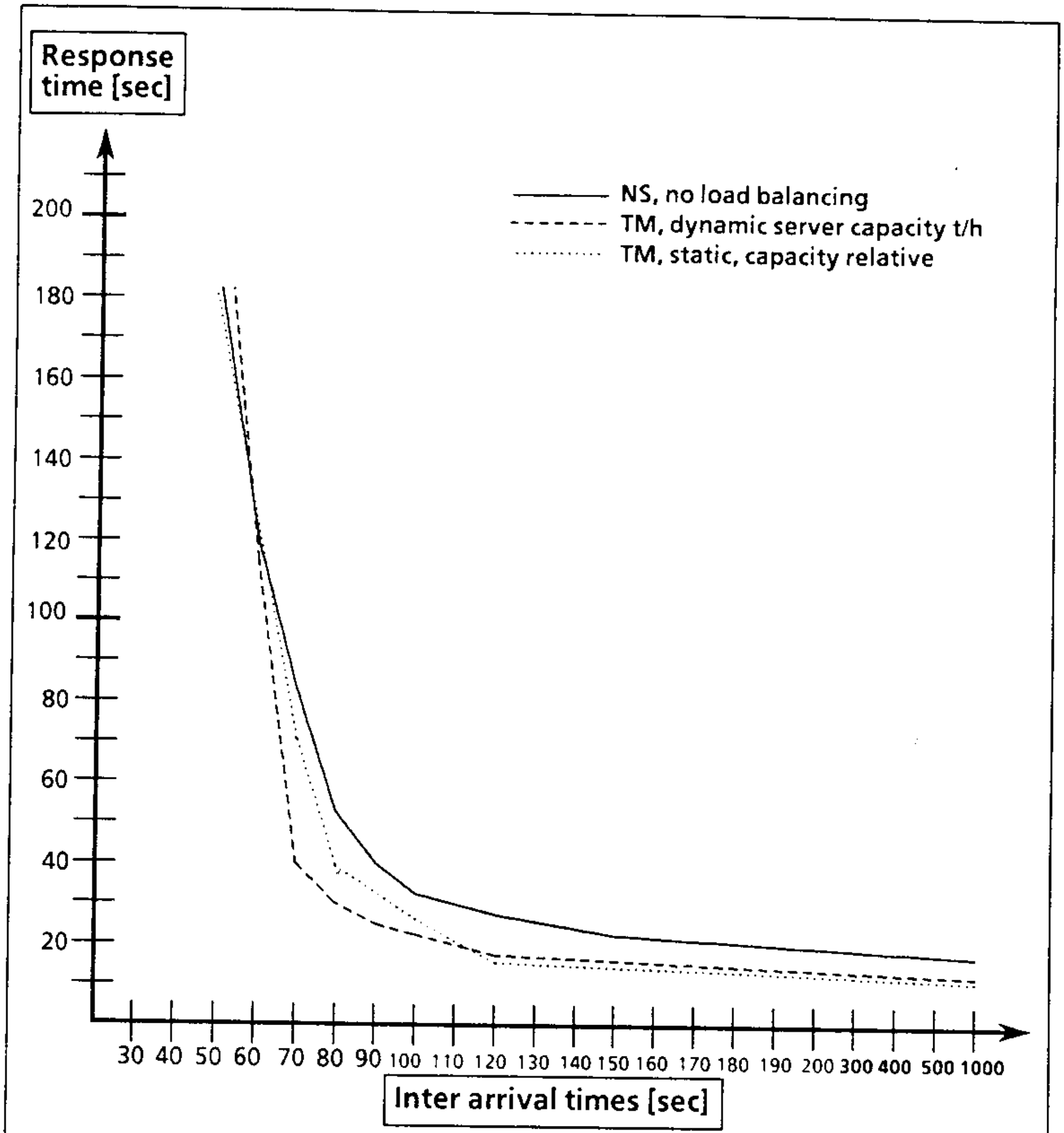


Figure 4.3.11: Effect of load balancing on response times

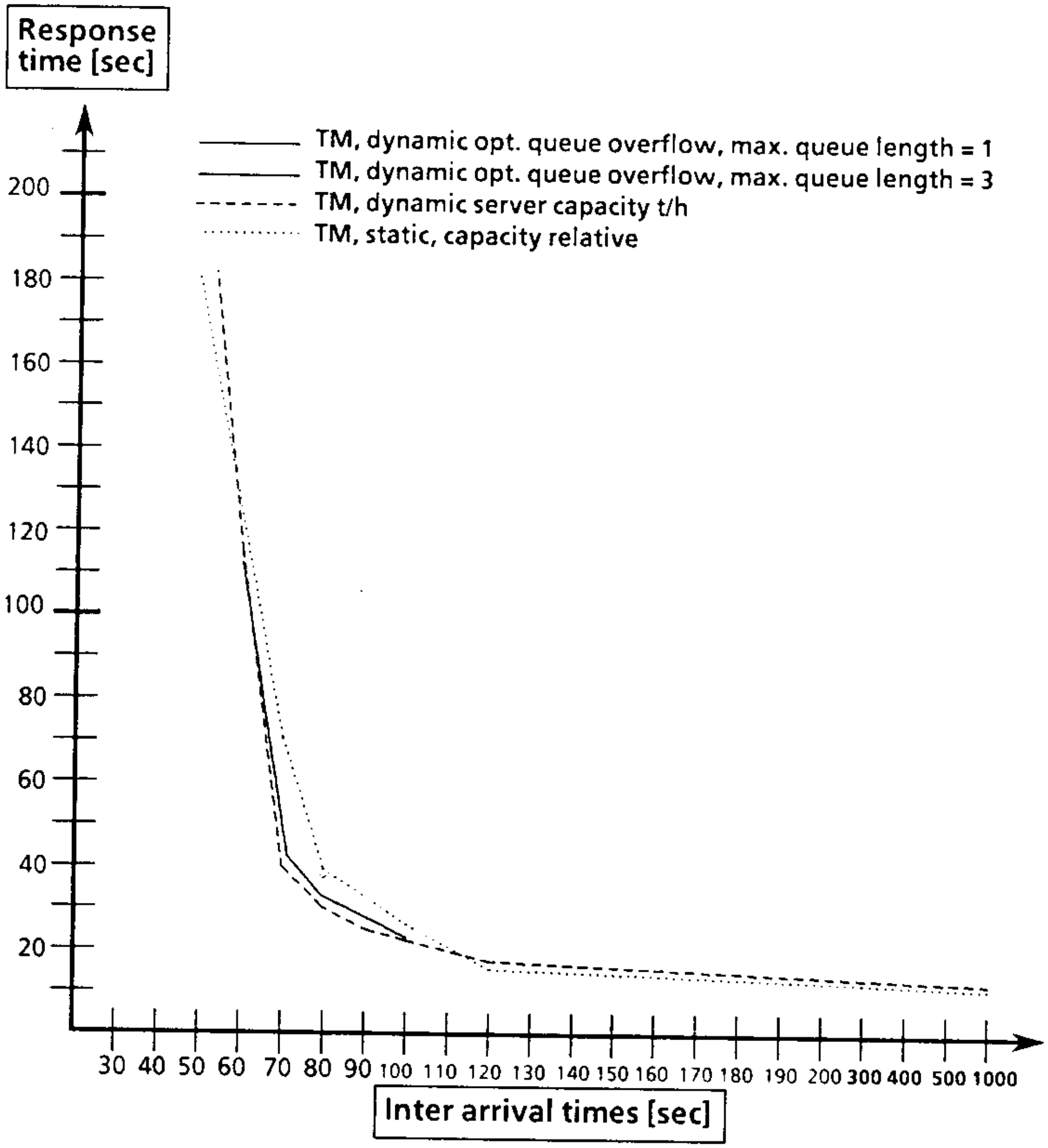


Figure 4.3.12: TM load balancing algorithms, a comparison

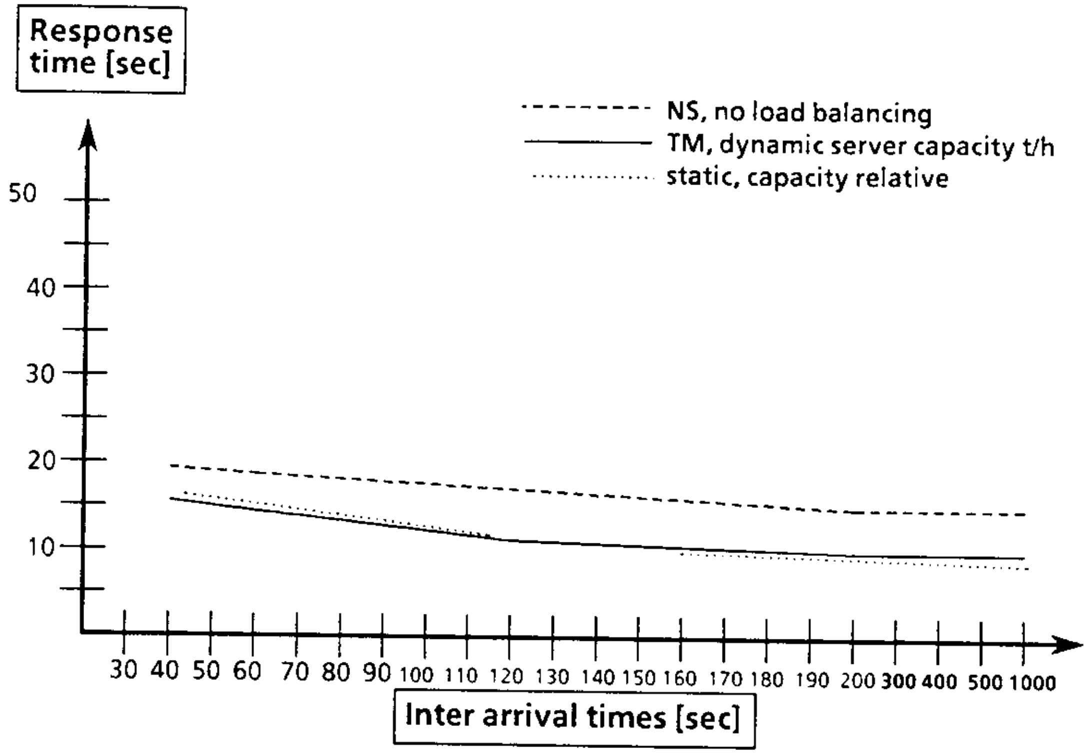


Figure 4.3.13: Effect of load balancing on response times, network speed increased x10

4.4.2.4 Effect of changing TM processing time requirements

Objective: Assess impact of varying TM processing times on R_{tm}.

Procedure: Run the TM simulator for varying TM processing times as defined in 4.4.1. Assume network speed is doubled to prevent it from being a bottleneck.

Results: As discussed in 4.4.1, the TM "processing penalty" or "load" has been simulated at +10%, +30% (the default load) and +40% with the results plotted in figure 4.3.14. At IAT = 100sec, $R_{tm}[10\%] = 13$sec, $R_{tm}[30\%] = 15$sec and $R_{tm}[40\%] = 16$sec. The workstation utilization did not meaningfully change as the TM processing requirements increased to 40%: it increased from max. of 0.26 at 30% to max. of 0.28 at 40%.

Analysis: As expected, the lowest R_{tm} is supported by TM load of 10% and the worst R_{tm} is associated with the 40% load. For IATs of 100 seconds and higher the response time reduction is directly related to the TM load therefore it should be implemented as efficiently as possible. Reducing the TM load from 30% to 10% will yield 2 seconds improvement (which is 12%). Since the workstation utilization is low, changing this parameter did not drastically affect R_{tm}.

Conclusions: The TM should be implemented efficiently but if 30% is unachievable, 40% will add only 1 second to the R_{tm} at IAT > 100sec, i.e. R_{tm} is not very sensitive to TM load changes.

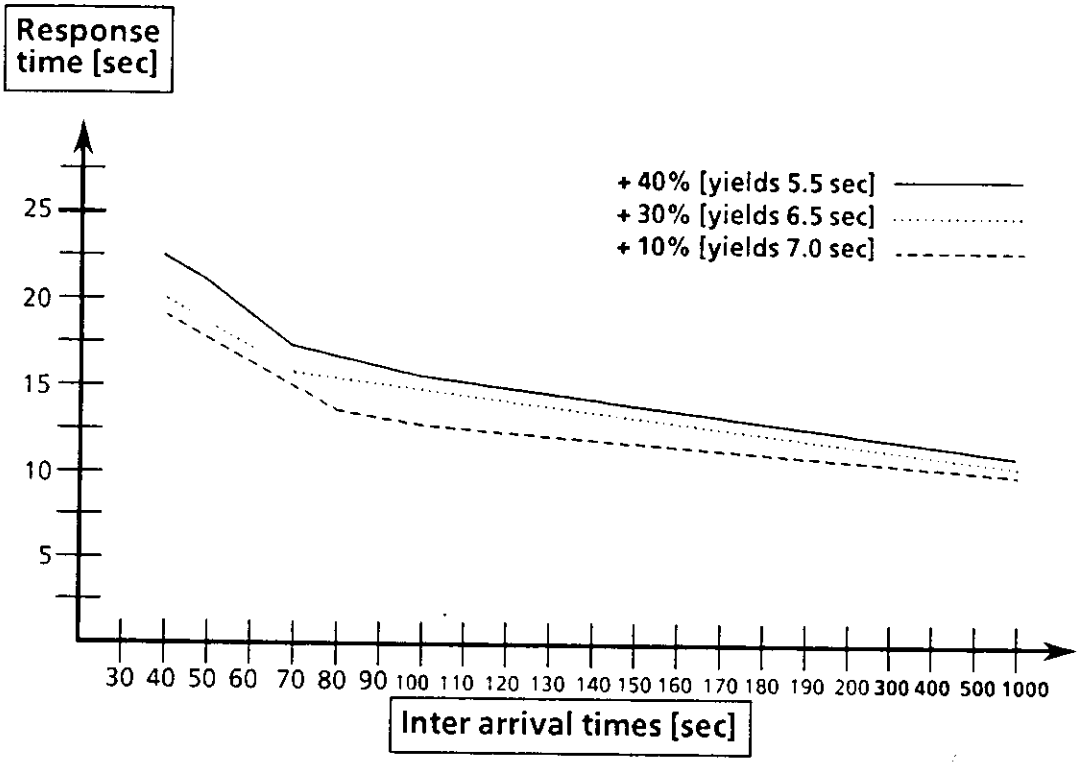


Figure 4.3.14: Response time sensitivity to TM processing requirements

4.4.2.5 Effect of changing workstation processing power

Objective: Assess impact of varying workstation processing power on R_{tm}.

Procedure: Repeat the simulations increasing workstation processing power by x2 and x5 for the TM and NS cases to provide for a comparison.

Results: The results are plotted in figure 4.3.15. Workstation utilizations which were very low have been lowered thus keeping the workstation utilization at less than 25% for all IATs simulated.

Analysis: As expected, R_{tm} is directly related to the workstation processing power at all ranges of IAT. We observe that R_{tm}[TM, workstation speed at x1]>R_{tm}[TM, workstation speed at x2]>R_{tm}[TM, workstation speed at x5] and similarly for the NS case.

However,

R_{tm}[TM, x1]<R_{tm}[NS, x2] and

R_{tm}[TM, x2]<R_{tm}[NS, x5].

Conclusions: (a) Due to **R_{tm}[TM, x1]<R_{tm}[NS, x2]**, it will be better to add the TM rather than spend money on doubling workstation processing speeds.

(b) Due to **R_{tm}[TM, x2]<R_{tm}[NS, x5]** doubling the workstation speed and adding the TM may prove more cost effective than just increasing the workstation processing power by x5.

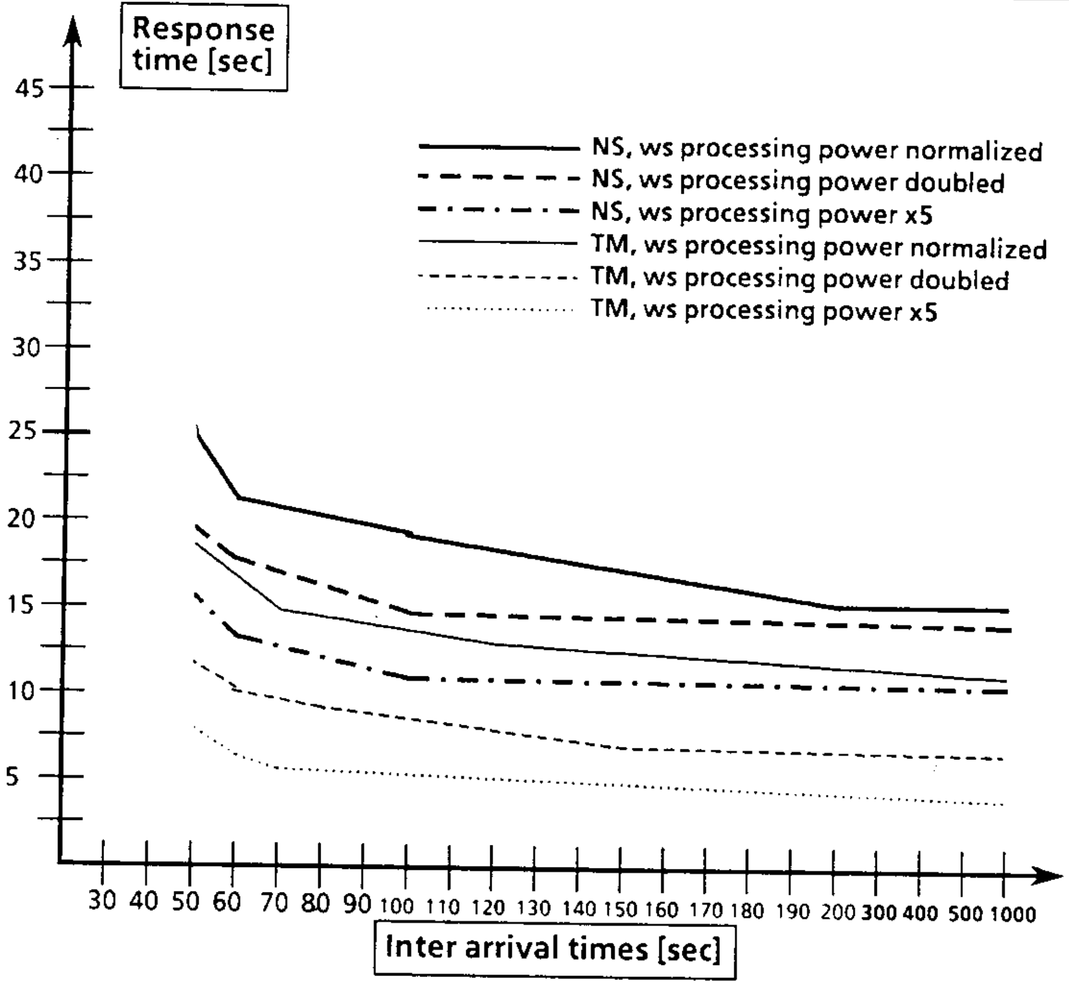


Figure 4.3.15: Effect of changing workstation processing power on response times, NS vs TM

4.4.2.6 Effect of changing system servers' processing power

Objective: Assess impact of varying servers' processing power on R_{tm}.

Procedure: Since the servers' utilization rate is very low (as discussed in 4.4.2.1), repeat the simulations for reduced servers' processing power by a factor of 10 with and without the TM to provide for a comparison.

Results: The results are plotted in figure 4.3.16. R_{tm}[NS] increased substantially, especially at IAT<90sec.

Analysis: At IAT = 100sec, R_{tm}[NS] increases from ~30seconds to ~50seconds (20 sec which is 66%) while R_{tm}[TM] increases from ~21seconds to ~29seconds (8 sec which is 38%). Clearly, this is due to the load balancing algorithm better utilizing the servers. The pattern repeats itself for higher IATs: at IAT = 150sec, R_{tm}[TM] increases by ~3seconds while R_{tm}[NS] increases by ~10 seconds.

Conclusions: In the system simulated, reducing the servers' processing speeds by a factor of 10 will increase the response times but reduce the system cost. If the TM is implemented and the addition of 3 seconds (@IAT = 150sec) is not meaningful, system costs could be reduced.

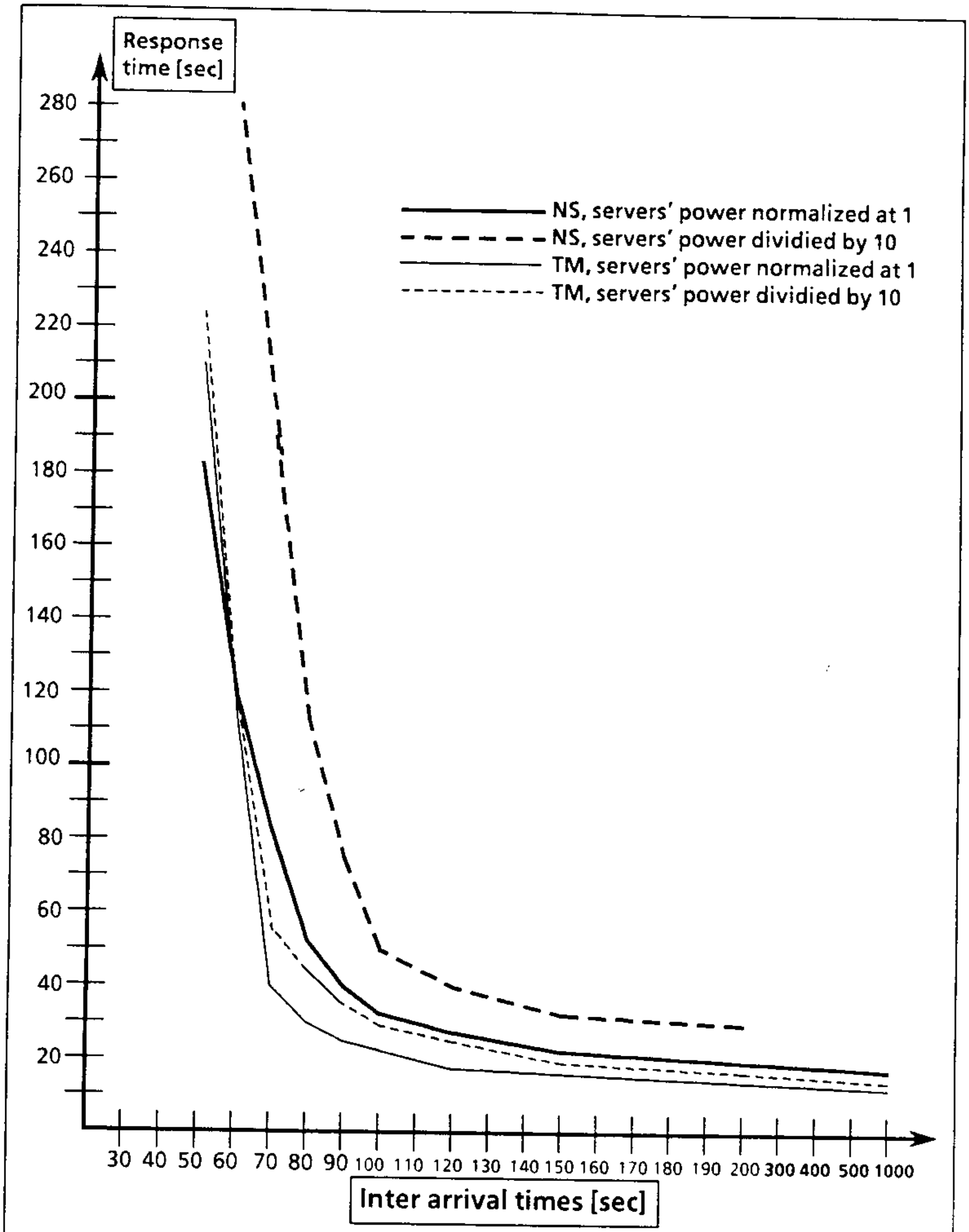


Figure 4.3.16: NS and TM response time sensitivity to reducing servers' processing power

4.4.2.7 Effect of changing job size on R_{tm}

Job size and job execution times are indicative of job "complexity".

Objective: Assess impact of varying job size on R_{tm}.

Procedure: Repeat the simulations for job sizes of 100KB ± 50% for the TM and NS.

Results: The results are plotted in figures 4.3.17 and 4.3.18. Figure 4.3.17 shows R_{tm} as a function of IAT while 4.3.18 plots the network utilization for the TM and NS cases @50KB/job and 150KB/job. Note that some curves show a sharp drops in the range of $200<IAT<300$ seconds due to the x-axis scale change. Network utilization for the TM case at 150KB/job is similar to the NS case except in the IAT range of $80<IAT<150$ seconds where the dotted line represents the TM curve.

R_{tm} increases as job size increases. It has also been observed that workstation and server utilization factors, which were very low as discussed before, remained very low. For example: in the TM case, job size = 150KBytes, at IAT = 90seconds the server utilization factors were: ser1: 1%, ser2: 3.8%, ser3: 0.9%, ser4: 4.3% and ser5: 4.7%.

Analysis: R_{tm}[job size = 150KB] increases steeply as IAT drops below ~150seconds due to the bottleneck already identified before: the network. Due to the large transmission times, the network starts affecting R_{tm} at lower arrival rates (higher IATs) compared with R_{tm} for 100 and 50KBytes. This analysis is supported by the utilization versus IAT curves plotted in figure 4.3.18. At 50KBytes the network does not create a bottleneck thus the TM's response times are better than NS's (attributed to load balancing).

Conclusions: By decreasing job size to 50% of the default, the TM yields better response times at all IAT ranges. If IAT is increased by 50%, the TM response time superiority over the NS starts at IAT = 150 seconds rather than 60 sec, i.e. 50% increase in job size changes the TM / NS break even point by 250%. Therefore, a decision whether to follow the TM concept depends upon the expected real life IAT and job size.

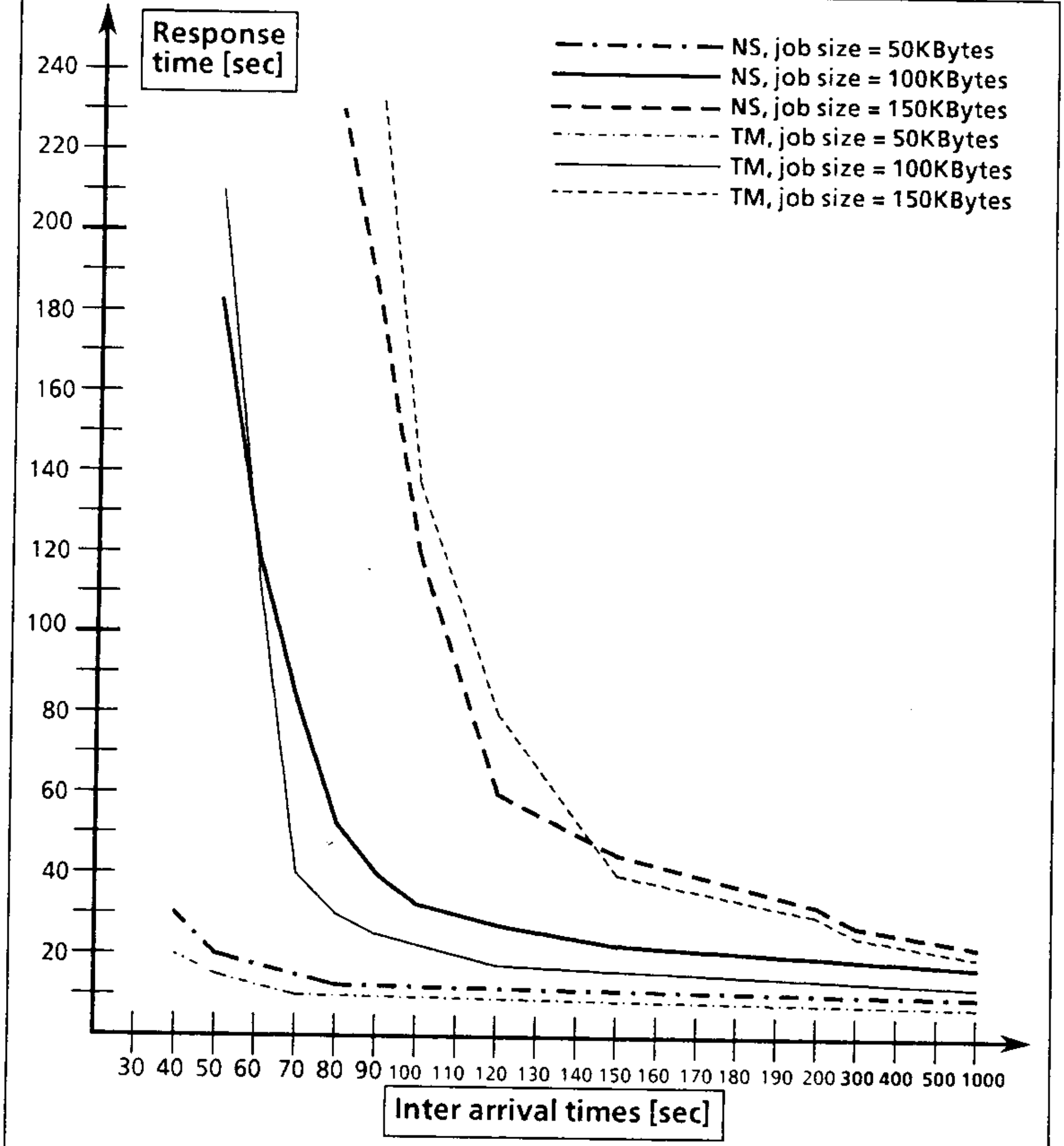


Figure 4.3.17: Response time sensitivity to varying job sizes

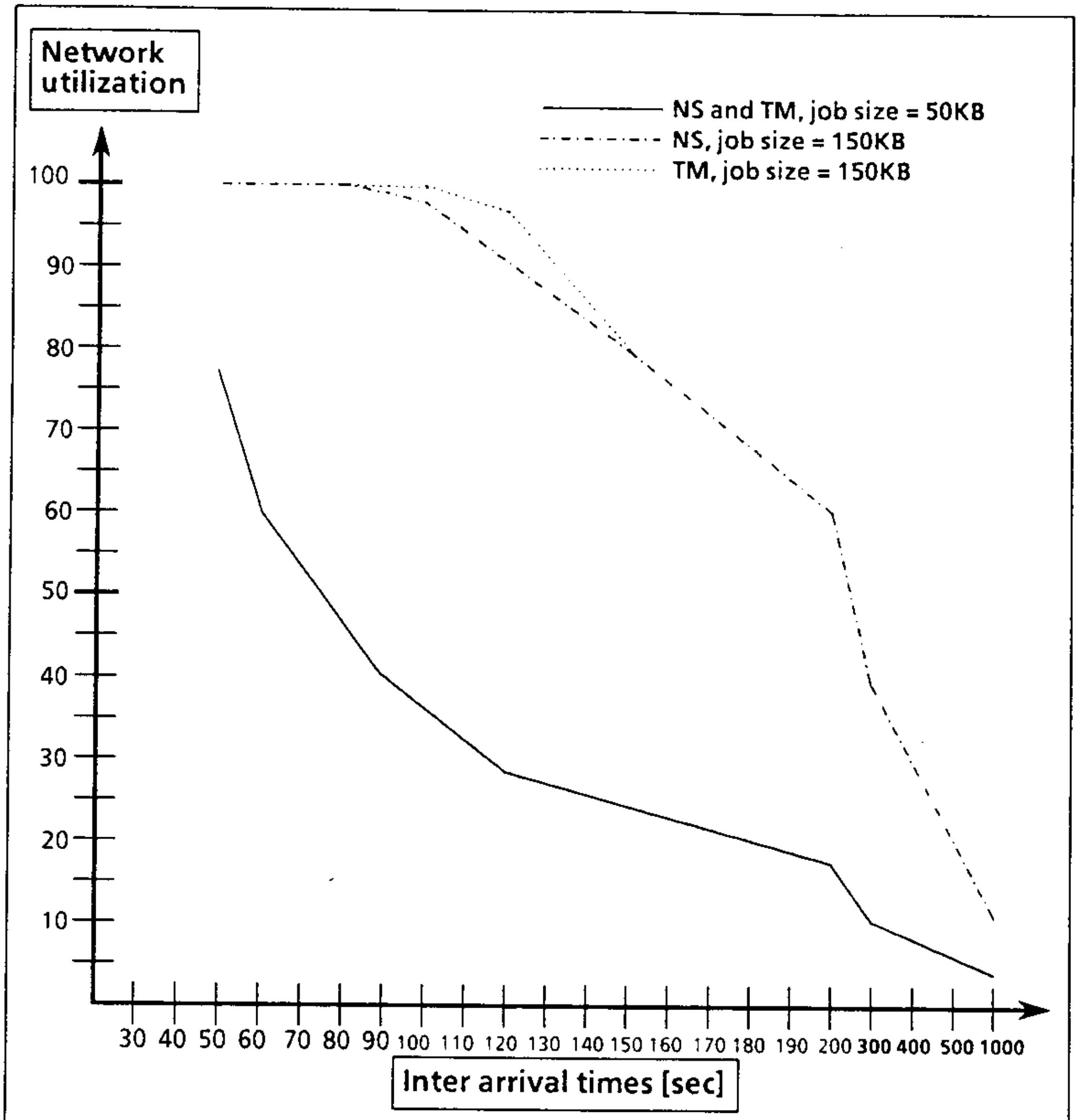


Figure 4.3.18: Network utilization @ various job sizes

4.4.2.8 Effect of changing job processing time

Job size and job execution times are indicative of job "complexity".

Objective: Assess impact of varying job processing time on the system response times.

Procedure: Repeat the simulations for the default job processing times ± 50% for the TM and NS. Plot R_{tm} versus IAT.

Results: The NS and TM results are plotted in figures 4.3.19 and 4.3.20 respectively. We observe that, as expected, with and without the TM, R_{tm} increases as processing time requirements increase. At IAT = 100sec, for the TM, increasing job processing time by 50% adds ~10 sec delay; decreasing it by 50% decreases R_{tm} by 5sec. As in 4.4.2.7, server utilization (UT), although slightly increased due to the increase in processing time, remained low at less than an average of 4% (calculated as follows: (UT_{max}[Ser1 for all IATs] + . . . + UT_{max}[Ser5 for all IATs]) / 5).

Notice that (1) the R_{tm} (TM versus NS) break even points for the default and + 50% processing times occur at IAT~60 seconds and (2) R_{tm}[NS, default processing times] = R_{tm}[TM, processing time increased 50%].

Analysis: The reason for (1) above is: changing processing time affects all of the system servers with the exception of the network since the network is sensitive to message length only. Thus the bottleneck will appear at the same IAT. However, as the job processing time increases, R_{tm} increases as well for all curves. Therefore, these curves do not provide additional information to support a tradeoff discussion of TM versus NS IAT crossover point. The curves become steep as IAT decreases due to the network creating a bottleneck.

Conclusions: Due to R_{tm}[NS, default processing times] = R_{tm}[TM, processing time increased 50%] a conclusion is made that adding the TM will facilitate increasing job processing time by up to 50% while preventing a response time increase. In fact the response time will improve as depicted by figures 4.3.19 and 4.3.20.

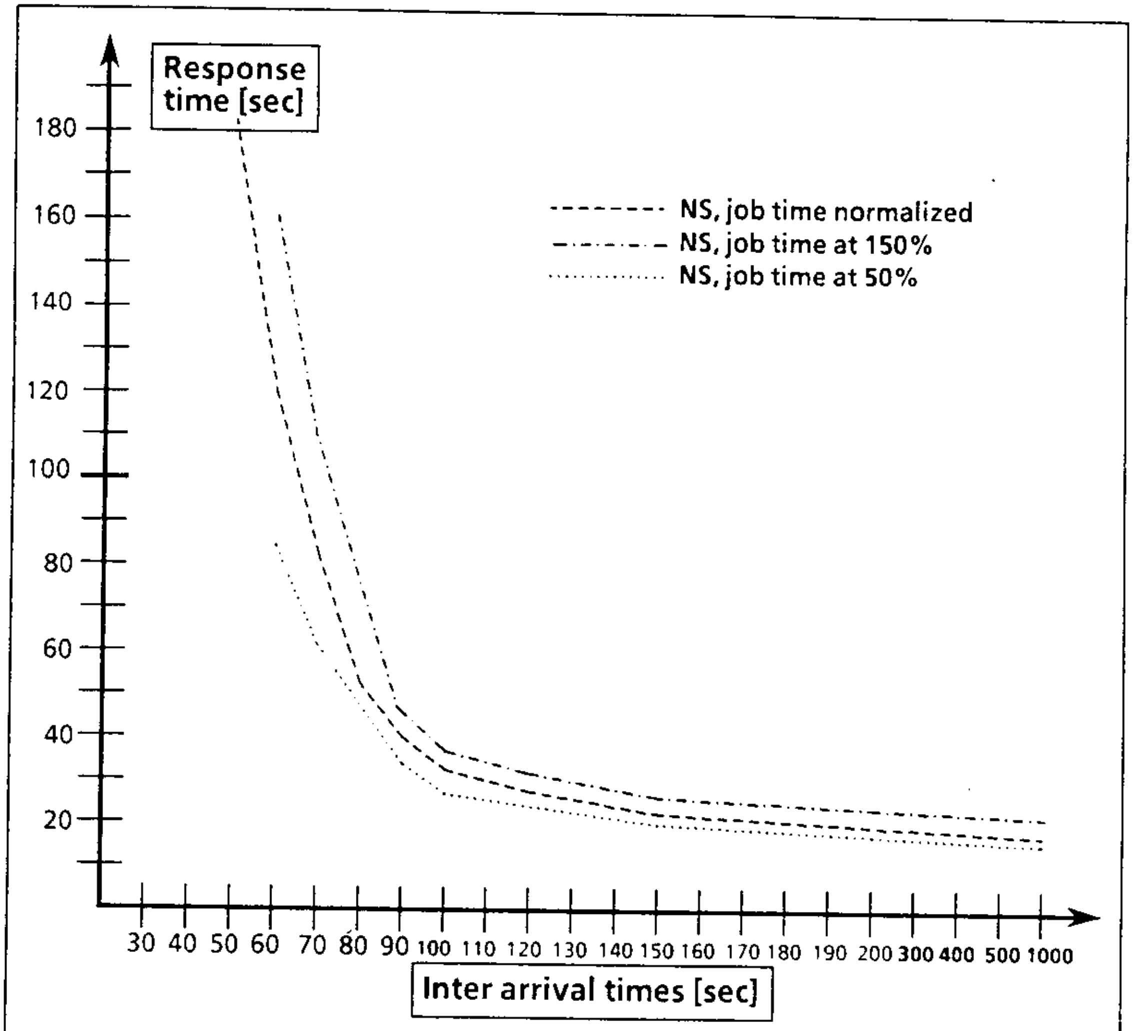


Figure 4.3.19: Effect of changing job processing times on R_{tm} / NS

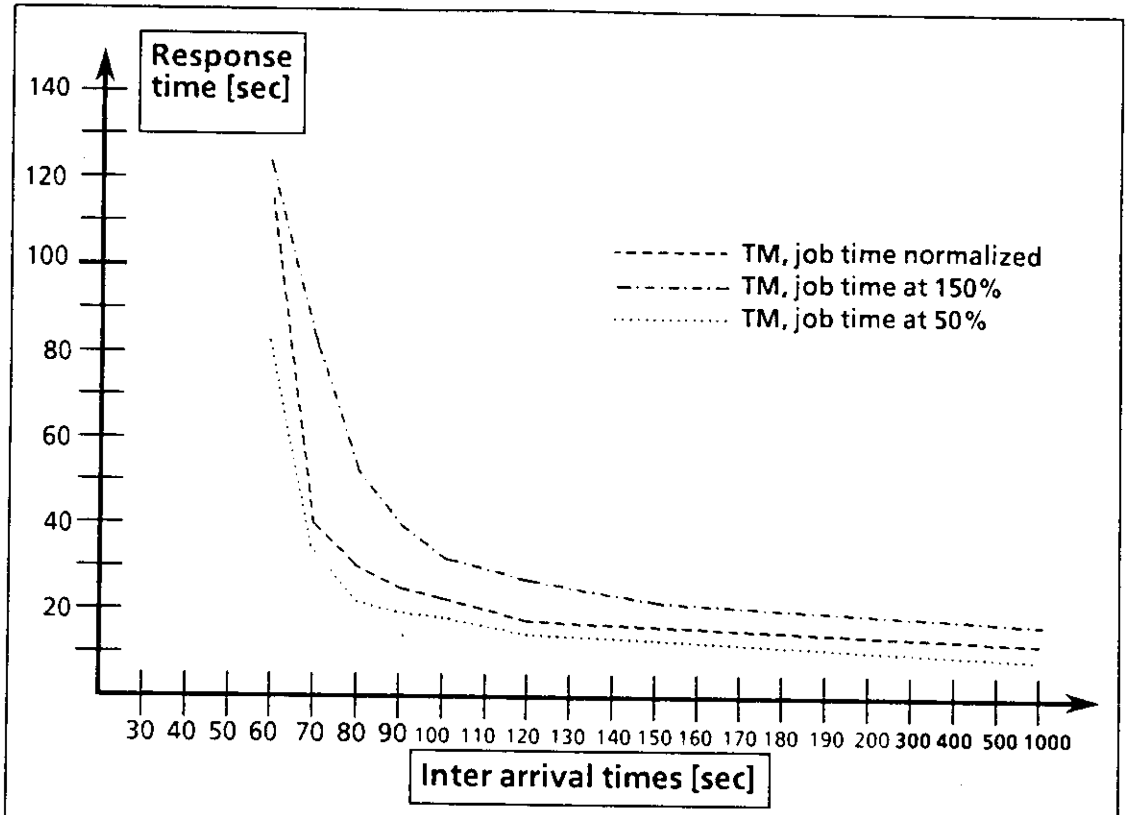


Figure 4.3.20: Effect of changing job processing times on R_{tm} / TM

4.4.2.9 Effect of changing user think time

Objective: Provide a measure to assess impact of varying user think times on the NS response times. This figure will help quantify response times improvement based on various user "speeds".

Procedure: Repeat the NS simulation used in 4.4.2.1 for the following user think times: 3, 6, 15 and 30 seconds.

Results: The results are plotted in figure 4.3.21. Note that the 6 seconds curve merges with the 3 second curve at IAT = 90sec. Server's utilization remained low. At IAT = 60 sec, and think time of 30 seconds, server 1 through server 5 utilizations were: 2%, 2.5%, 7.8%, 3%, and 13% respectively.

Conclusions: If user think times are greater than the 3 second default time (which very likely is the real life case) than the TM response time improvements are greater than previously discussed. The improvement can be quantitatively sized using figure 4.3.21.

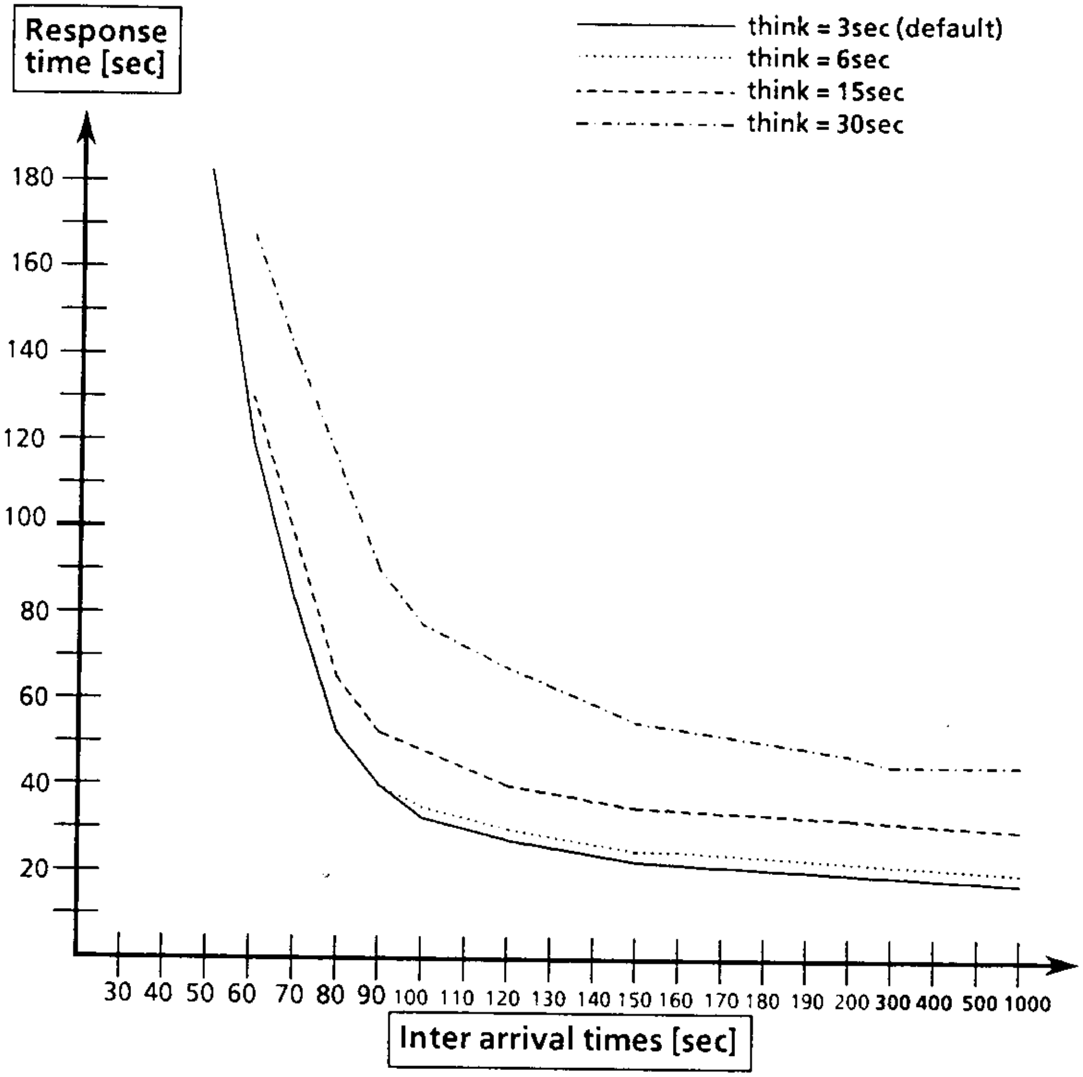


Figure 4.3.21: R_{tm} as a function of IAT for given user think times

4.4.2.10 Centralized TM

In this case, there is a server which is fully dedicated to performing the TM function. It does not perform any other processing.

Objective: Provide an answer to the following question: for the default parameters, how does the centralized TM compare against the distributed TM.

Procedure: Simulate a centralized TM assuming it is fully dedicated to the TM functions. Its processing power is equivalent to the sum of the 32 workstations' processing power dedicated to the TM. Since the TM function -or TM transaction- has been executed in 1.5 seconds per workstation, and knowing that there are 32 workstations, the TM's processing power has been fixed at 0.05seconds per TM transaction (1.5 seconds divided by 32). See 4.4.1 and 4.4.2.4. The TM processing power has also been increased 10 times to 0.005 seconds per TM transaction processing.

Results: R_{tm} versus IAT and network utilization are plotted in figure 4.3.22. The centralized TM response times for 0.05 and 0.005 seconds per TM transaction are less desirable than the distributed TM. TM utilization was less than 4.8% at all times.

Analysis: Since the workstations in the simulation are cooperative, the TM assigns tasks back to them with the same probability function as in the distributed case. Due to job transmission from the TM to the servers and to the workstations, the network creates a severe bottleneck as shown in figure 4.3.22.

Conclusions:

- With the default parameters, the distributed TM provides better response times. Note that the centralized TM has many drawbacks as described in 4.3, one of them is the need to a dedicated site which increases cost. Fault tolerance has to be considered as well.
- The TM did not introduce a bottleneck.

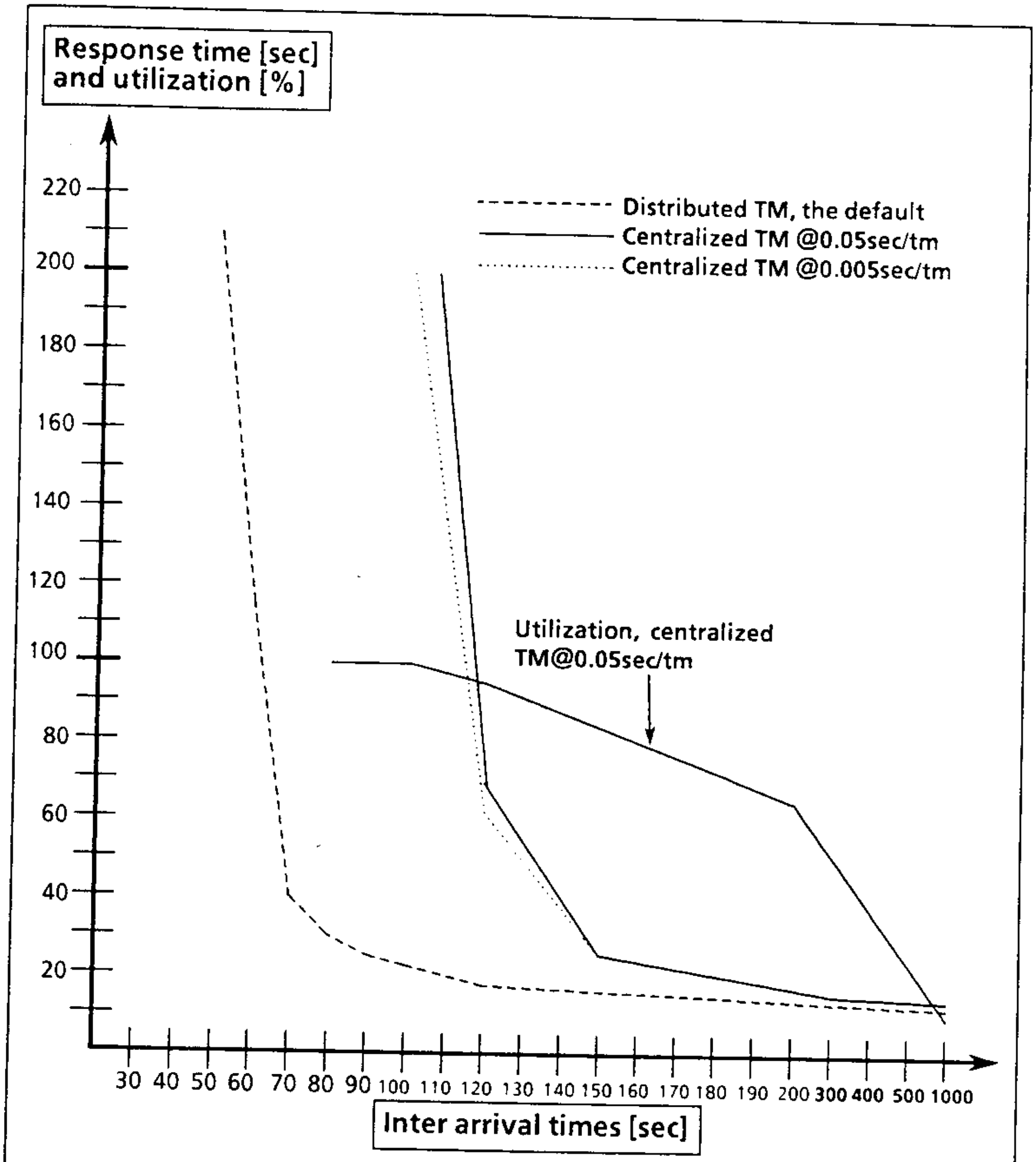


Figure 4.3.22: Centralized TM response time

Simulation notes

RESQ was run until a confidence level of at least 94% was achieved. Then min., max., and average values have been obtained. All curves have been plotted on or between the minimum and maximum data points. In some cases, 4.4.2.3 for example, the results' resolution did not facilitate a clear distinction between two adjacent curves.

At some high arrival rates ('high' depends on the actual simulation but in most cases 'high' roughly means IAT<70 seconds) the number of arrivals exceeded the number of departures. The spread between the min. and max. values given by RESQ led to a quoted confidence level of less than 85%. It was impossible to increase the simulation time since RESQ run out of virtual memory space. Nevertheless, RESQ, using Little's result, estimated the response time for the simulation duration. The lower confidence level in this IAT range has been accepted since our assumption, as noted in 4.4.2.1, has been that real life IATs will not be this stressful and in fact be greater than 100 seconds inter arrival time. The theoretical network overhead calculation performed in 4.4.2.3 supports the simulation and shows that arrivals exceed departures at IAT of roughly 70sec.

Lastly, the simulation has a deficiency since an assumption was made regarding the workstations: local user processing does not affect the workstation utilization, which may not be always true.

In summary: The most important conclusion is that the TM concept has proved useful for the real life range of job arrival rates. It cuts response times, as seen by the user, considerably (by up to 50 percent). The assumption that network utilization will be low has been proved wrong. The network, based on the stated assumption, will be a bottleneck, unless its speed increased at least by 2.

4.5 XNS Simulation

This section describes the Ethernet physical and link levels simulation followed by the higher layers in the XNS model. The simulation was done in SLAM and run on a VAX 11/750 (VAX is a Digital Equipment trade mark). The purpose of this simulation was to supply throughput parameters to the Application simulation described in the previous section and answer questions (a) through (c) stated below. For no extra efforts, the simulator provided other information like Ethernet collision rates.

Since we are studying the XNS environment with the idea of using it as a basis for further enhancements, we were interested in understanding the throughput versus utilization and collision rate, and the effect of changing packet size on throughput. If with current servers and users the network is "fully utilized", adding the TM with extra servers may create problems: low throughput, high collision rate, limited expandability and loss of interactivity. In this system, individual workstations (or nodes) acquire, process, and dispense files around the network, seizing and releasing communication bandwidth on the Ethernet in an equal access, random manner.

Major differences between the TM approach and traditional office systems are:

Network Traffic: The TM may manage voice mailboxes, scanners, printers, workstations, optical filing systems, magnetic filing systems, voice servers as well as other image manipulation servers for still or moving video dealing with raster images. In many cases, these images will be routed automatically from node to node at machine decision rates (milliseconds) rather than user rates (seconds) thus creating large bursts of file transfers.

File Size: Due to the above, files can only be expected to get larger as indicated in section 4.4. It is expected that most files will reach 100KBytes in length.

As a result the offered network load will increase while only slightly increasing the number of Ethernet drops per network. Nevertheless, office systems with or without the TM as described in this work share a common feature: Ethernet and XNS. Therefore, the questions asked are:

(a). Will packet collisions increase to such a point that virtually all packets are destroyed and nobody can gain access?

(b). Will throughput fall off badly so that users lose interactivity?. Is the Ethernet a bottleneck? Should fiber optic LANS be recommended?

(c). How sensitive is throughput to the number of workstations and packet size?

To answer these questions we started with the Ethernet simulation and continued with the higher level protocols in XNS.

4.5.1 Ethernet Simulation Model

The Ethernet model is shown in figure 4.4. The model is limited to the Data Link and Physical layers of the XNS Ethernet protocol as defined in [Eth82]. No attempt was made to include workstation parameters. Packets attempt to acquire the network or follow standard back-off procedures for collisions, and are destroyed after they have seized the net. Two or more packets trying to size the Ethernet at the same time signal a collision. This is a simulation of a single send-receive pair of nodes. Window size (number of packets per acknowledge) is ignored since it is irrelevant at this level. The model is an exact representation of what happens when a packet is communicated from one node to another from the point of view of the Ethernet. Network utilization can be controlled directly by altering the inter-arrival time of packets. We used an existing model, fed it with our updated parameters, and ran multiple trials. Simulated were 576 byte packets, and "Blue Book" [Eth82] values for all other parameters. Figure 4.5 charts the results.

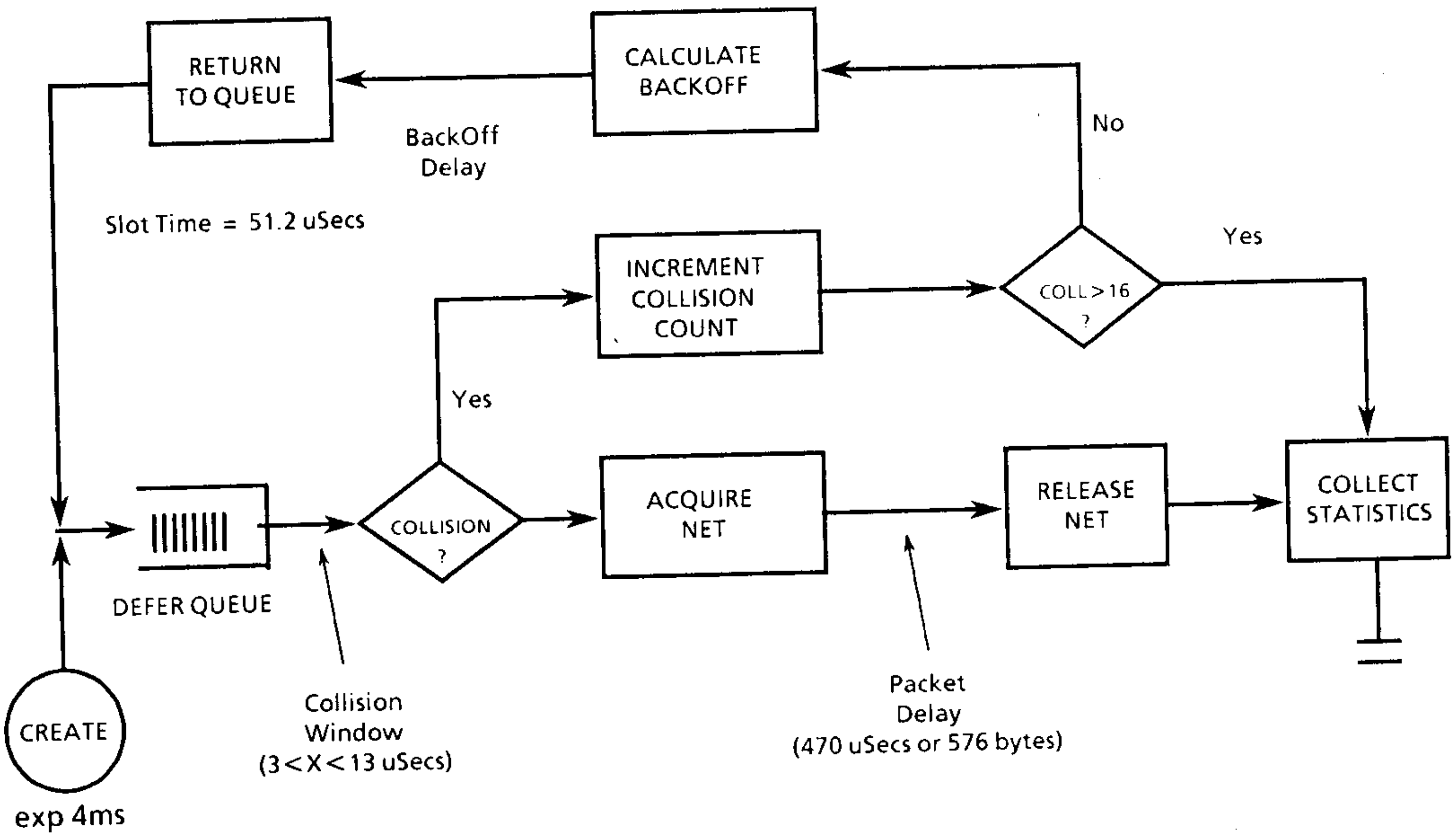


Figure 4.4: Ethernet model diagram

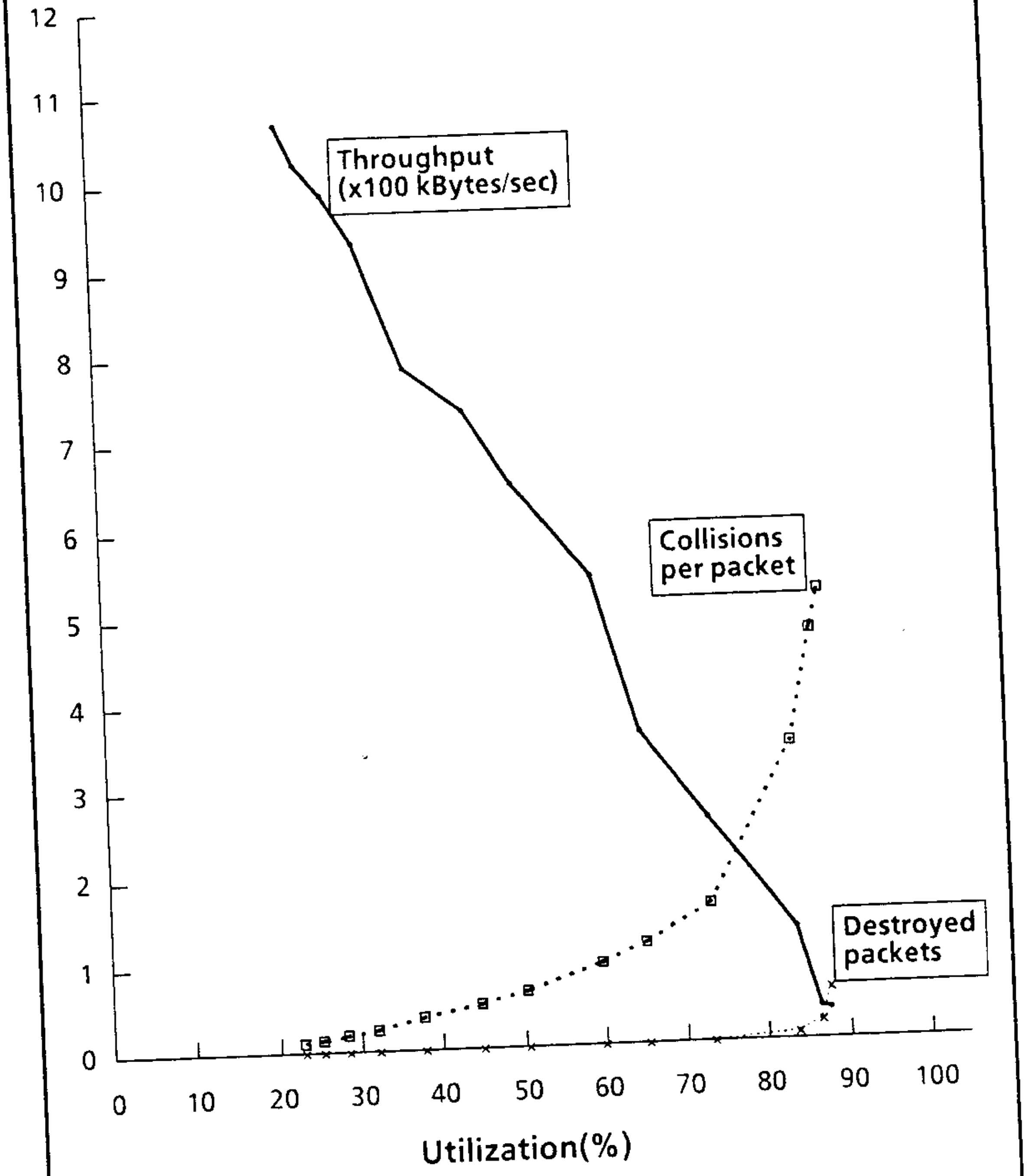


Figure 4.5: ENET model results at 576 byte packets

Several conclusions were achieved: The network remains stable (meaning throughput doesn't drop sharply) even under heavy utilization. The collision rate increases dramatically only above 70% network utilization. Throughput falls about 100 kBytes/sec for every 10% increase in utilization. This relationship held constant even above the 70% limit. Based on these results and knowing that current network utilization is less than 5% we concluded that the Ethernet may not be a bottleneck and may be adequate to support additional traffic.

4.5.2 The XNS Model

The following model has been developed based on existing simulation packages in order to simulate the higher layers of XNS. It includes two components: the Ethernet model discussed before (XNS level 0) and empirical processing overhead imposed by XNS layers 1, 2, and 3 (Courier only) as measured on the Xerox 6085 work station. Packet transmission error rate of 10^{-7} has been included. There was a need to simulate the effect of varying the number of workstations contending for the Ethernet as well as the effect of a multi file transfer including inter-frame and inter-file delays as imposed by the 6085. This XNS model is depicted in figure 4.6. The dotted lines in the figure do not mean closed loop population but rather Courier remote procedure calls to prompt transmission of the next frame and file. Inter arrival times due to the XNS protocols are also indicated.

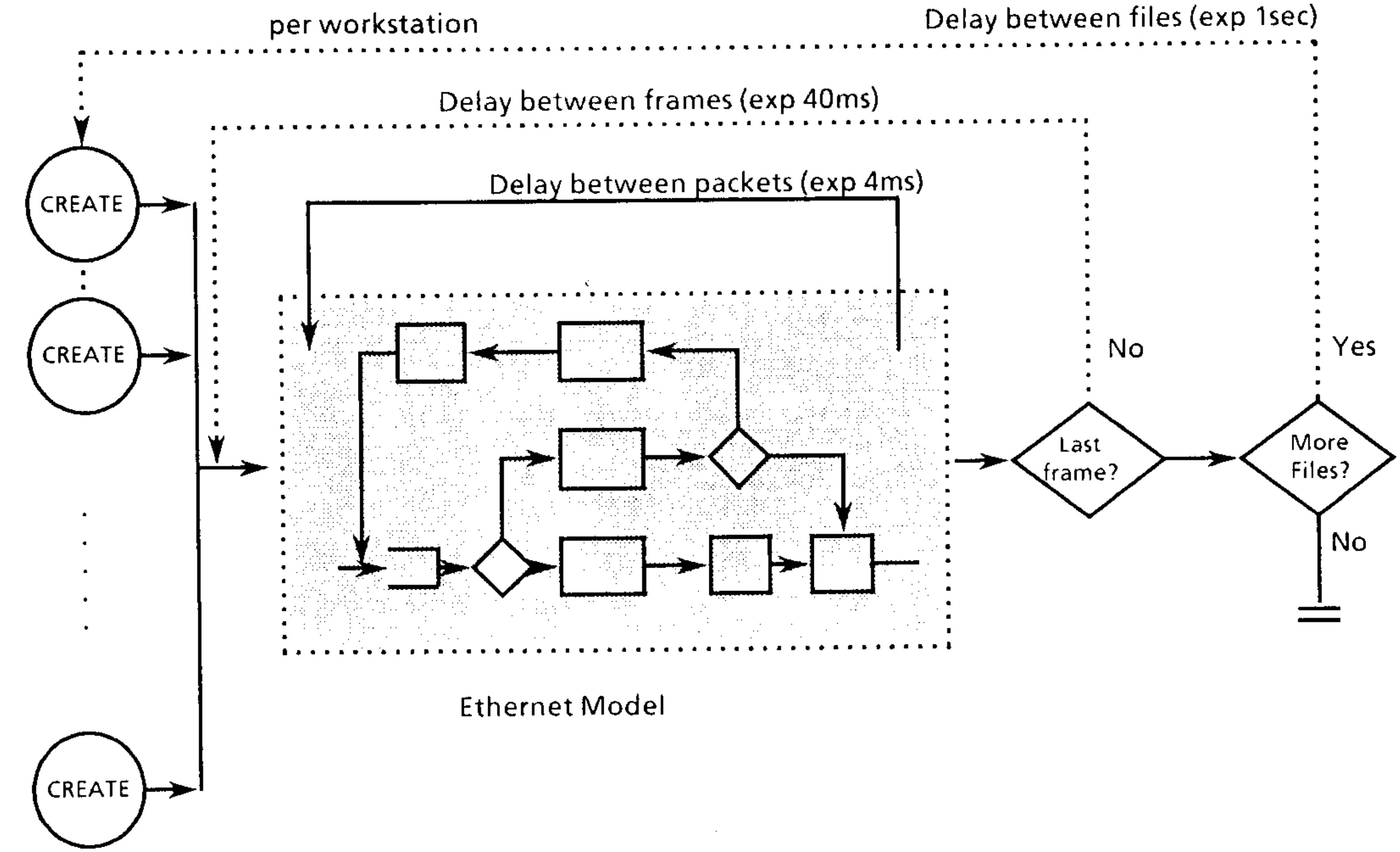


Figure 4.6: The XNS model

Before proceeding with the model, we clarify the following: *Offered Load* is defined as the average aggregate number of packets/sec offered to the network. Each workstation freely generates a certain number of packets per second given file size, file IAT, and data packet size regardless of the network capacity. It is important to note that destroyed packets are also included.

The model was defined by a combination of fixed and variable parameters. The fixed parameters were: **File size** - a typical data file in this system: 100KBytes. **Maximum number of workstations** - the maximum number of sending nodes in the simulation: 32. **Inter packet spacing** - the hardware recovery time before the next packet could be sent: not less than 9.6 usecs. **Slot time** - the unit of retransmission time used in the backoff algorithm: 51.2 usecs. **Minimum & maximum collision round trip time** - the time required for a round trip on the Ethernet which is the time required to ascertain there was no collision on the network: 3 to 13 usecs. **Network bandwidth:** 10 MHz. The variable parameters were: packet length in bytes, number of sending nodes, number of packets in frame (packets per ack), IAT between packets in frame, number of frames in a file, IAT between frames, and IAT between files.

The simulation provided the following packet information: creation time, collision count, originating node ID, packet sequence in frame, frame sequence in file, and file creation time & ID as well as:

IAT of packet creation. The distribution of times between packets on each workstation. This gave us a clear indication of network fairness.

Histogram of packet collisions. A large number indicates that each node is spending much time deferring; a low number indicates good throughput.

Packet lifetime. The backoff algorithm itself enforces a binary exponential waiting distribution. Packet lifetimes will show this wait and we concluded that variations in individual lifetimes were not significant.

IAT of frames and *network utilization*.

File lifetime. A clear indication of throughput (the min. and max. numbers amplify the network equality issue).

4.5.2.1 XNS Simulation Results and Analysis

The simulation objective was to measure effective throughput and network utilization as a function of packet size, offered load, and number of workstations. As mentioned earlier, we also wanted to watch access fairness. For each run, a specific packet size, number of workstations, and offered load were selected. Each run was repeated 10 times changing only the random number seed in the SLAM processor.

Three packet sizes: 576, 1500, and 4096 bytes per packet were studied. The first size corresponds to the current and only maximum packet length used. The second size is the XNS maximum allowed in the XNS protocols. The last size is used as a reference point to possible performance gains. The number of workstations was selected from 1,2,4,8,12,16,20,24,28, or 32. File IAT ranged from 4 seconds per file to 0 seconds which is the highest possible.

Figure 4.7 depicts the network utilization for each of the three packet sizes as a function of offered load. The ideal case (network utilization equal to offered load) is shown by the dotted line. For all three packet sizes, the ideal case is closely approximated until 50% offered load has been reached. At that point, the curves begin to diverge from the ideal case and from each other until the network utilization is virtually flat. Nowhere did we observe a collapse in network communication indicated by a falling network utilization.

Figures 4.8 through 4.12 show effective throughput rates as a function of number of workstations for ranges of offered loads and packet sizes. Not all graphs have "complete" curves: some network loadings were impossible to achieve with too many or too few workstations and the file IAT distribution selected. Several observations were made. For 576 byte packets, the number of workstations is not a critical parameter. The effective throughput ranged from 70,000 bytes per second down to 48,000. Additionally, offered load changes make no real difference on the throughput. For 1500 byte packets there is a moderate impact from node count but throughput is slightly affected by offered load. Large packet sizes appear to be most affected by node count and offered load.

4.5.2.2 XNS Simulation Conclusions

Earlier in this section we raised 3 questions. The first was: "will packet collisions increase to such a point that virtually all packets are destroyed and nobody can gain access?" Provided that we do not exceed 50% network utilization, the results of the Ethernet the XNS simulation gave an indication that the network is 'stable' under heavy loads and the collision count is very low as indicated in figure 4.5.

The second question ("will throughput fall off badly so that users lose interactivity? Is the Ethernet a bottleneck? Should fiber optic LANS be recommended?") has been answered as well. Throughput will not deteriorate badly as depicted in figures 4.8 through 4.12 and interactivity will be maintained provided we use the smaller packet size of 576 bytes per packet. (We must also accept the fact that larger packet sizes imply significantly better throughput but throughput will fall sharply as network loading exceeds 50%. At the same time, it is also clear that throughput falls off considerably as node count increases).

The third question ("How sensitive is throughput to the number of workstations and packet size?") is also answered. For packet sizes of 576 and 1500 bytes, the sensitivity is low, as depicted in figures 4.8 through 4.12. Using 4KBytes/packet changes the answer dramatically.

Figure 4.7: Network Utilization vs Offered Load

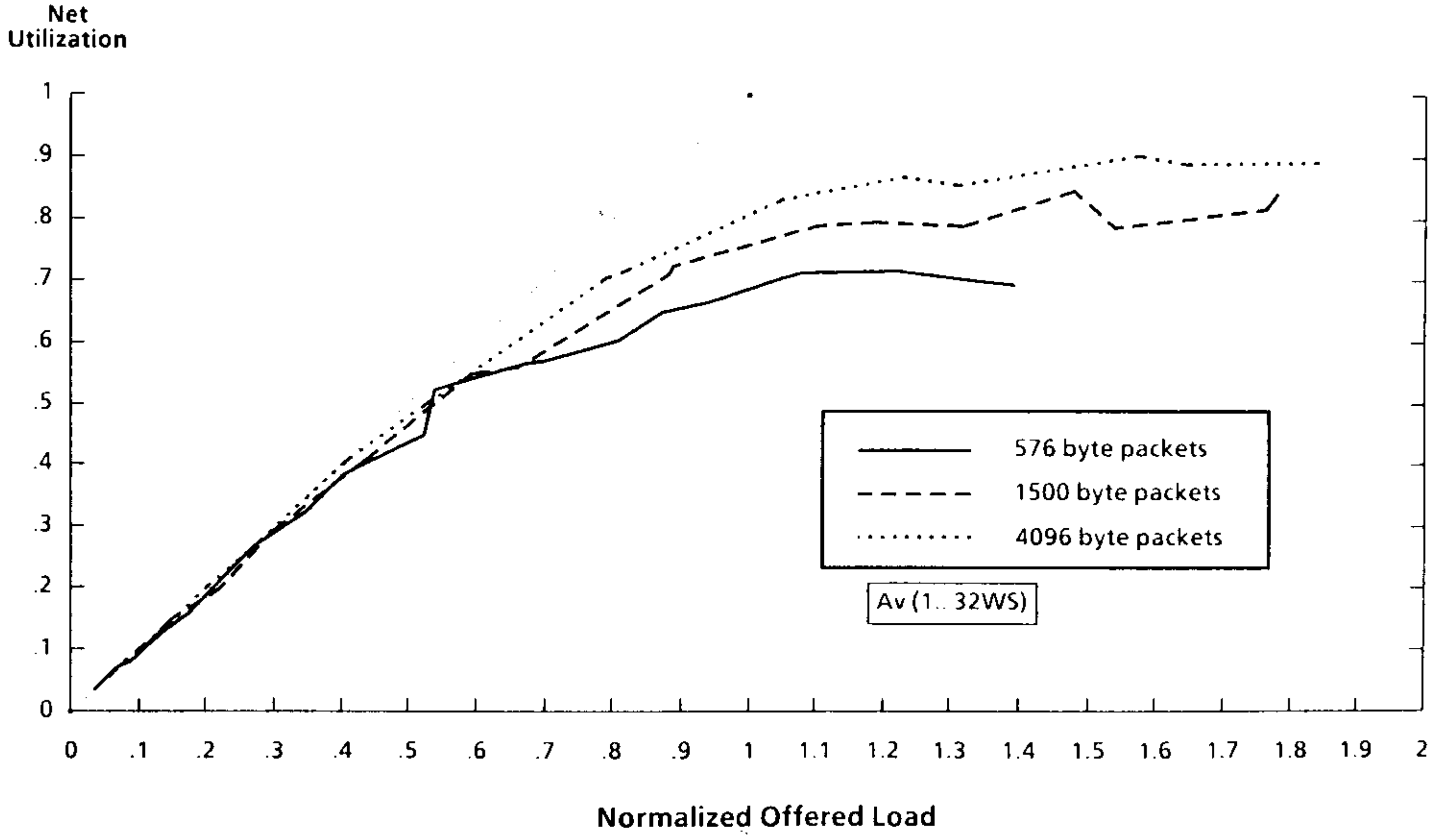


Figure 4.8: Effective Throughput vs Workstation Count
(offered load<25%)

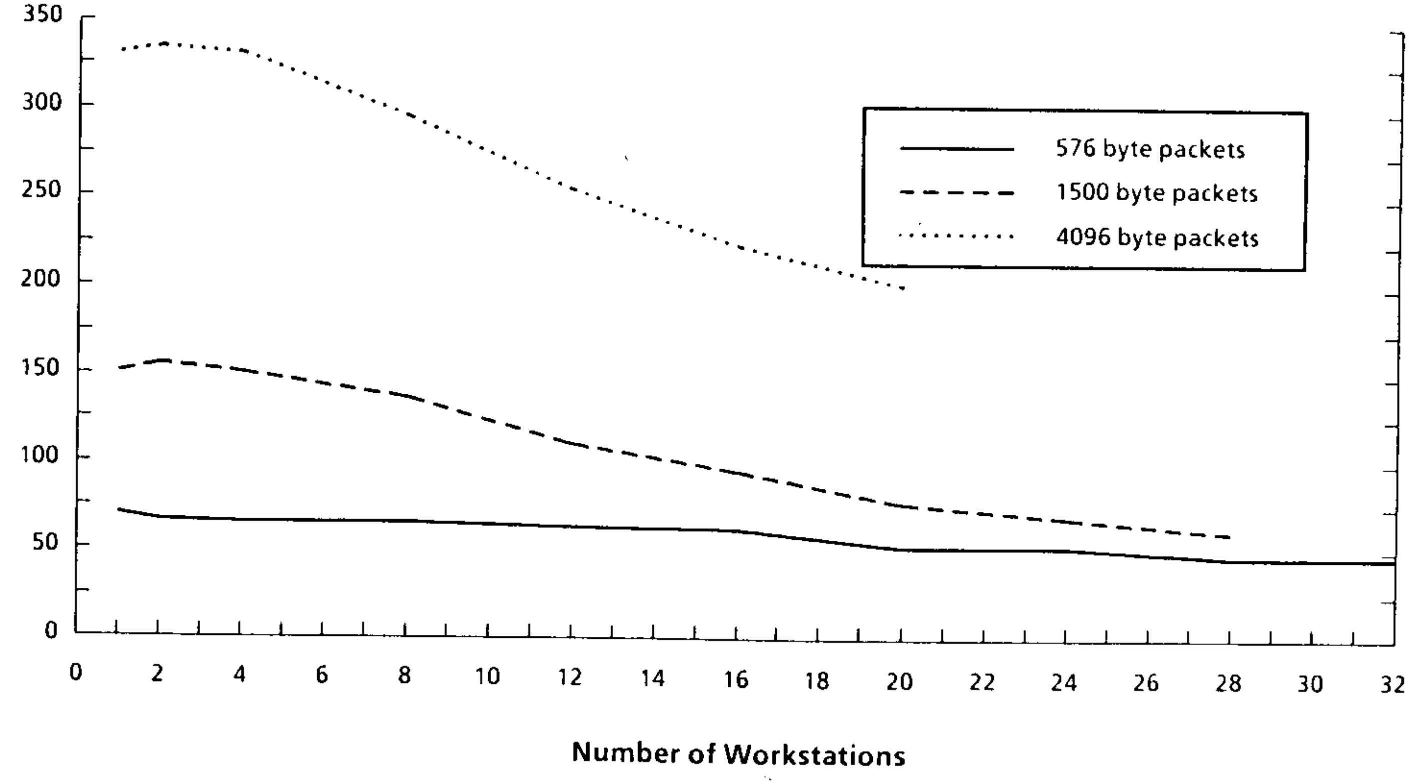


Figure 4.9: Effective Throughput vs Workstation Count
(50% >offered load>25%)

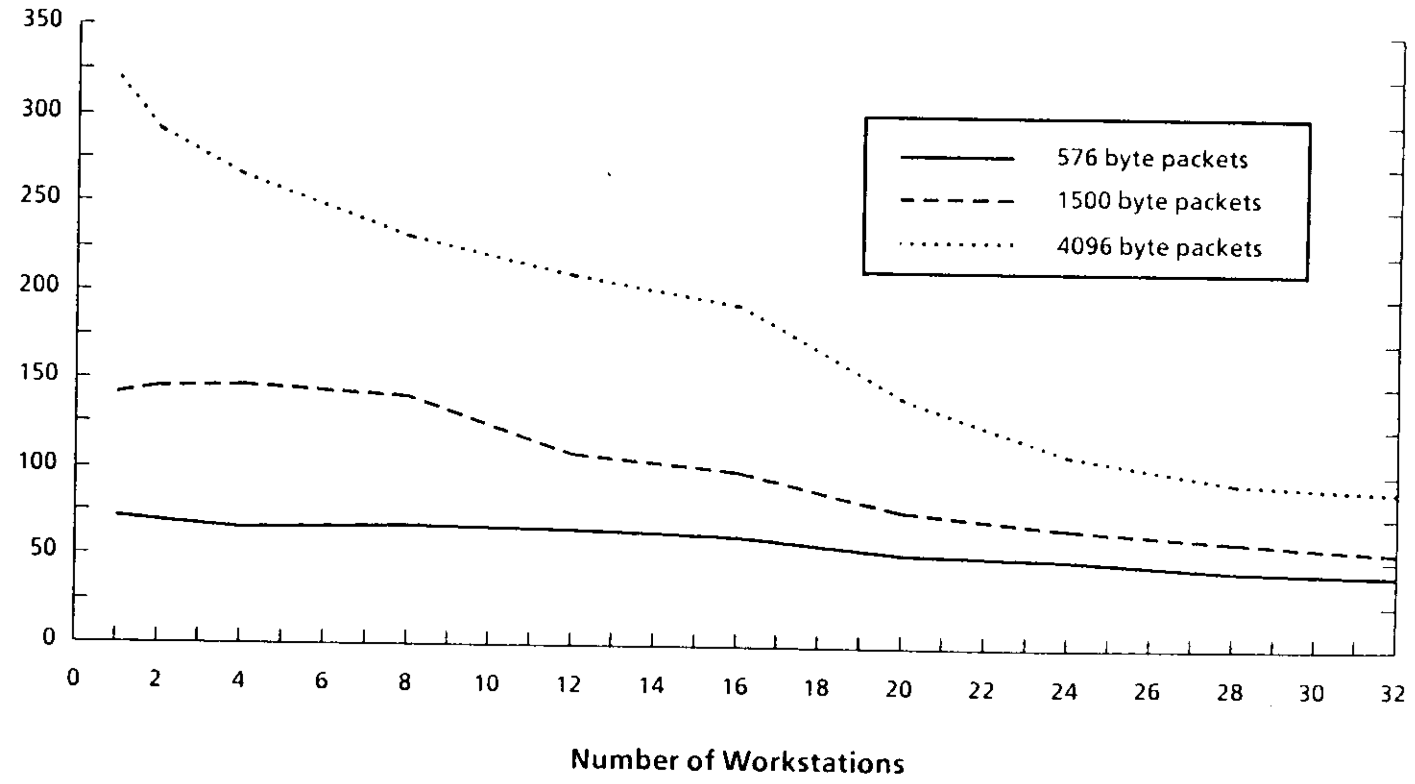


Figure 4.10: Effective Throughput vs Workstation Count
(75% >offered load>50%)

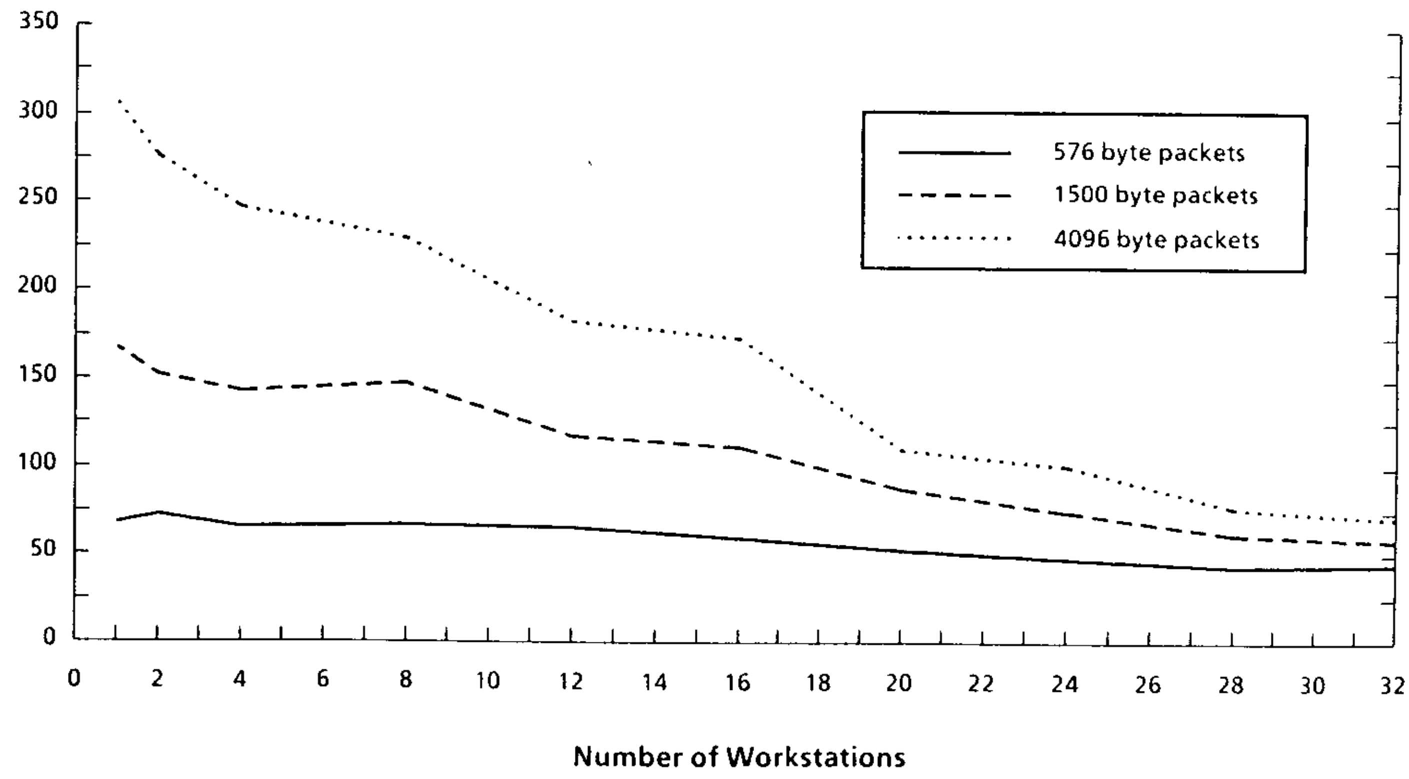


Figure 4.11: Effective Throughput vs Workstation Count
(100% > offered load > 75%)

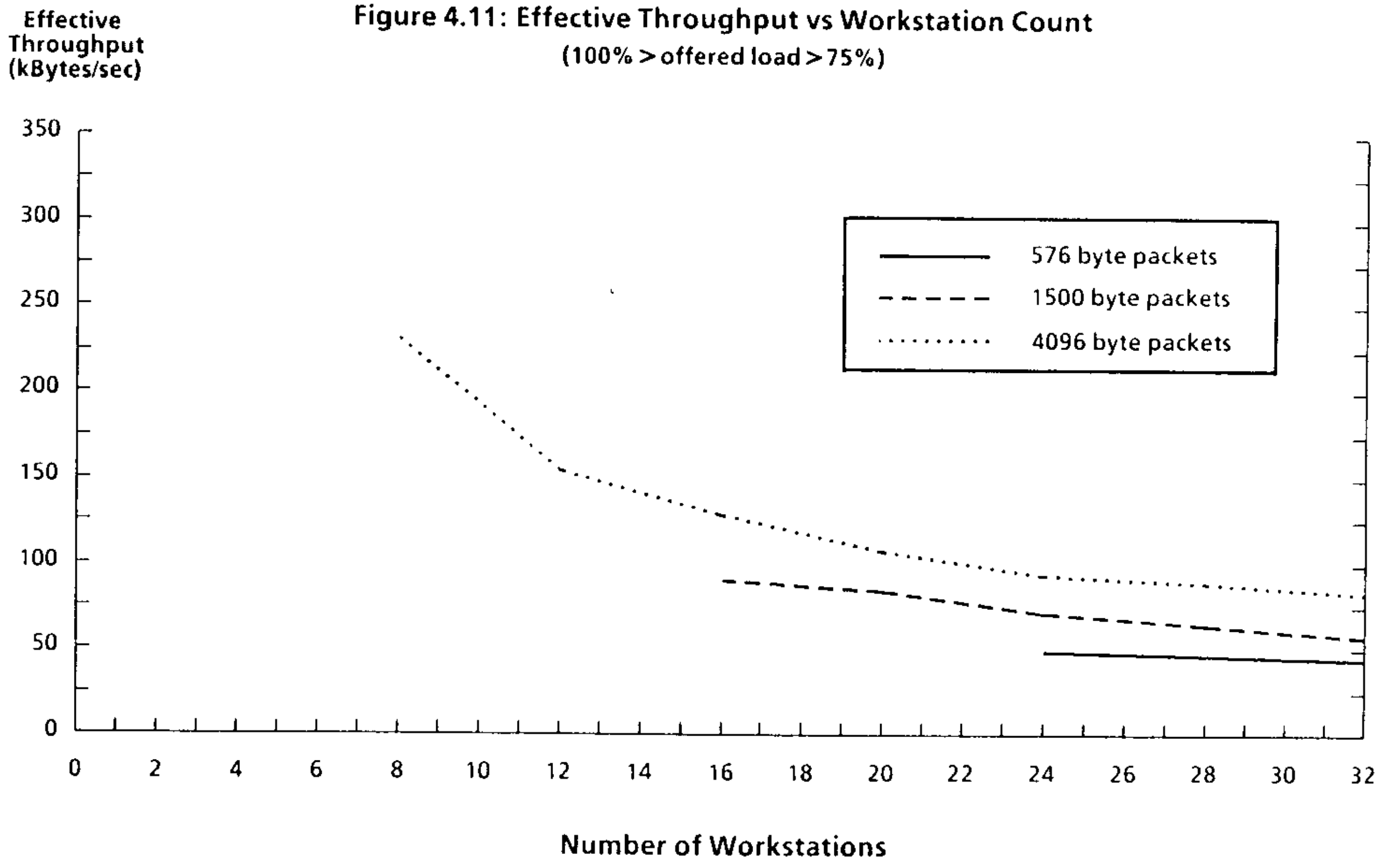
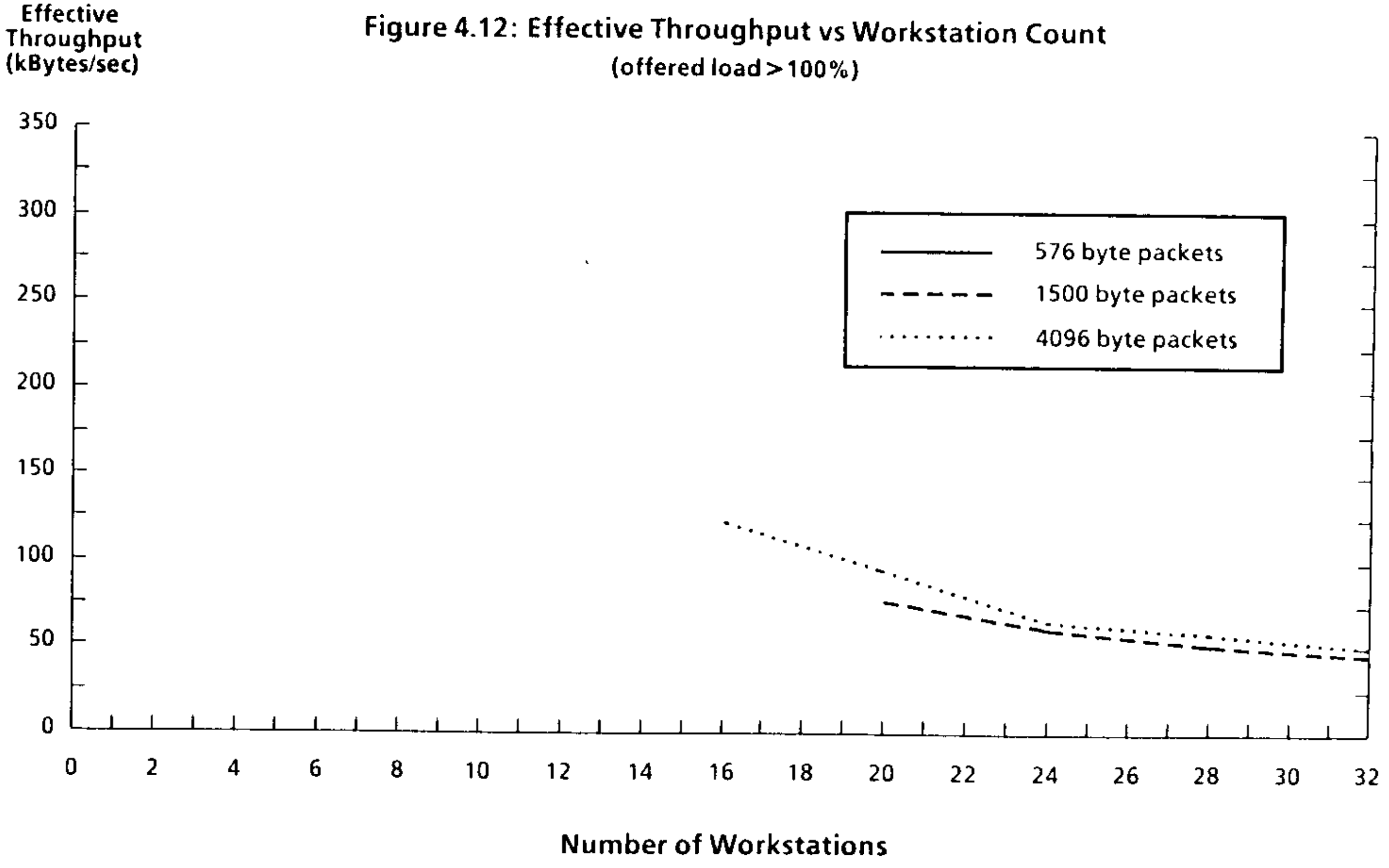


Figure 4.12: Effective Throughput vs Workstation Count
(offered load > 100%)



5.0 Conclusions and Future Research

A major conclusion supported by the simulation results shown in fig. 4.3.4 is the fact that the introduction of the TM improves response times by at least 25%. It is evident that the TM, although slightly increasing network traffic, reduces the parameter that users are most sensitive to: response time. The response time is reduced from 19 to 14 seconds for typical arrival rates of inter arrival times (IAT) greater than 500sec. The communication overhead introduced by the dynamic load balancing algorithm does not nullify the TM's response time improvements. The experiments have also shown that the TM improves the response time even with static load balancing. Note that the hypothesis to be tested in this work was that the addition of the TM, despite its inherent processing and possible communication overhead, would improve the response time.

A series of ten experiment sets has been conducted to investigate the TM behavior in the specified simulation environment. The effect of changing the following features/parameters has been investigated: communication network transmission speed, various load balancing algorithms, TM processing time requirements, workstation processing power, system servers' processing power, job size and processing requirements, user think times, and centralized versus distributed TM.

For the specific parameters chosen in our simulation environment, the communication network turns out to be a bottleneck. Multiplying the network speed by 100 or even 10 is not justified since multiplying it by 5 eliminates the bottleneck anyway. In small systems (8 users) the network does not create a bottleneck therefore the TM is effective in increasing performance for any arrival rate. Considering response time, it has been shown that just adding the TM provides better response times than doubling the network speed. Furthermore, it is better to double the network speed and add the TM than multiply the network speed by 5 without adding the TM. It may also be more cost effective.

If load balancing is removed, response time consistently degrades in the entire IAT range. The server overflow algorithm performs even worse. Provided the network is not a bottleneck, the server capacity threshold load balancing algorithm is the scheme of choice for IAT<130sec and will perform very well at IAT>130 sec. As also confirmed by the analytic model, the communication overhead imposed by dynamic load balancing is minimal (at IAT>100sec). However, in the simulation, dynamic load balancing does not perform well at IAT<70sec since the network is fully utilized and imposes high delays due to the required status packets. The static server capacity algorithm provided the same response time as the dynamic server capacity threshold at IAT>200 seconds. Since implementing the static algorithm is simpler than the dynamic one, the static algorithm is recommended.

It has also been shown that the addition of the TM yields better response times than doubling the workstation processing speed. Furthermore, a combination of doubled workstation speed and the TM provides better response times than increasing the workstation processing power by x5. It may also prove more cost effective. Studying the effect of varying job complexity provided the following: If job size is increased by 50%, the TM response time superiority over the existing system starts at IAT = 150 seconds rather than 60 sec. If job size is decreased by 50%, the TM improves response times at all IAT ranges. Adding the TM will facilitate increasing job processing time by up to 50% while preventing a response time increase. In fact the response time will improve as depicted in figures 4.3.19 and 4.3.20.

The TM should be implemented efficiently. However, response time is not very sensitive to TM load changes (section 4.4.2.4). In the system simulated, reducing the servers' processing speeds by a factor of 10 will increase the response times but reduce the system cost. If the TM is implemented, response time will increase by 3 seconds (@IAT = 150sec) compared with 12 seconds without it. Therefore the TM facilitates better system utilization and enables cost cutting. Yet an additional conclusion is that if the network designers had a fixed amount of money to spend, it would have been wiser to spend less on servers and more on the communication network and workstations. Last but not least, the distributed TM provides better response times than the centralized TM.

Further research:

This dissertation addresses the office, engineering, and scientific environment with its set of servers, workstations and users. However, the environment may reflect a more heterogeneous set of servers, e.g. workstations represented by IBM PCs as well as SUNs, computing servers represented by the traditional departmental processors (e.g. VAXs) as well as super computers like CRAY etc. With this range of processing speeds, should TMs be tailored as front ends only? Can a general purpose TM perform well in this network as well or should it be restricted to a class of servers? A totally different environment is the factory with intelligent robots, laths, testing machines etc. In factories, the 'jobs' require several processing steps performed at different sites (e.g. raw material inspection robot, assembly machine, and final test) that may be managed by the TM. This environment, in fact, imposes the use of third party data transfers, where the 'data' is actually the physical product. How well is the TM suited for this type of jobs?

Another topic which also supports the environment mentioned above is researching the TM fault tolerance characteristics and developing reliability models. The TM is capable of detecting server failures and reassigning the tasks to alternative servers which are capable of performing the tasks. In addition, the TM itself may fail thus requiring TM failure detection and reassignment as proposed in chapter 3. These procedures including TM failure detection and assignment algorithms need to be further studied. In the office and scientific environments server (and TM) failure rates are relatively low and the effect of failures, in most cases, is not catastrophic. The failure rates of mechanical equipment (or servers) in the factories may be different and a single node failure may lead to stopping the entire production line. Other systems may also require high reliability in addition to performing the TM functions in real time and under time constraints.

In very large systems, there might be a need to implement several TMs. The TMs support a multitude of servers and users, perhaps in a mixture of homogeneous and heterogeneous environments. This case raises several questions: should the TMs be implemented in some type of hierarchy or should they all be at the same level, is it appropriate to mix front end TMs and general purpose TMs, should TMs be controlled by other TMs. Some design and performance issues may be further investigated: the number of TMs per system, number and type of users per TM etc. TMs may need to communicate with each other. Should the proposed communication proposal be modified? Resource management over large distances may imply relatively long delay times between the TM and servers. The TM adaptation to this and possibly other wide area network constraints may be studied.

The TM concepts may be expanded to include features required by large information systems. It may address distributed data bases which are composed of multi media as well as multi format components. In these systems, a query may be submitted to the TM; the TM, using its local data base and the UDF (user define file) may optimize the query by searching the databases which are the most likely ones to provide the needed information, sort and filter the information before responding to the user while supporting transparency. If needed, concurrency control algorithms should be incorporated in the TM.

Other research topics include the following: In this work a major assumption has been that the workstations are cooperative: the TM may assign tasks to network servers and to the originating workstation. Therefore, the effect of non-cooperating workstations -when the TM should assign tasks only to its servers- may be studied. A different topic is associated with the TM domain *increase* and *decrease* rules which affect the domain size. Increase and decrease rules may be further developed to support a more heterogeneous environment. Inter TM communication needs as well as security issues may be studied. The existing TM mechanisms described here provide support for these incremental needs.

There is also opportunity for further research by actually implementing this model in a local area network. It may provide a test bed for further investigation of the TM's properties and in fact may complement the simulation results by providing actual real life measurements. The effect of static and dynamic load sharing algorithms as well as TM failures may be studied. In my work, an assumption was made that servers do not fail thus an important TM property has not been simulated. TM domain management algorithms may be tested to address the effectiveness of *increase* and *decrease* rules.

6.0 Appendix: Application Level Simulation Code

```
/*
        This model was used for the TM simulation. The XNS simulator was
        similar but with several changes.

*/

MODEL: tm06NOV
METHOD:simulation
DISTRIBUTION PARAMETERS: arv
DISTRIBUTION IDENTIFIERS:think thinkz mess con
 THINK:exponential(1)
 THINKZ:exponential(0.001)
 MESS:exponential(6.5)
 CON:0.05
 MAX JV:2
 QUEUE:ser1
  TYPE:fcfs
  CLASS LIST:cser1
   SERVICE TIMES:1
 QUEUE:ser2
  TYPE:fcfs
  CLASS LIST:cser2
   SERVICE TIMES:2
 QUEUE:ser3
  TYPE:fcfs
  CLASS LIST:cser3
   SERVICE TIMES:4
 QUEUE:ser4
  TYPE:fcfs
  CLASS LIST:cser4
   SERVICE TIMES:0.1
 QUEUE:ser5
  TYPE:fcfs
  CLASS LIST:cser5
   SERVICE TIMES:0.5
 QUEUE:net
  TYPE:fcfs
  CLASS LIST:cnet dnet    messx snet
   SERVICE TIMES:4/3 4/3    4/600 4/600

 QUEUE:human1
```

```
  TYPE:fcfs
  CLASS LIST:cthink1 athink1
    SERVICE TIMES:think thinkz
 QUEUE:ws1
  TYPE:fcfs
  CLASS LIST:mess1 con1
    SERVICE TIMES:mess con

 QUEUE:human2
  TYPE:fcfs
  CLASS LIST:cthink2 athink2
    SERVICE TIMES:think thinkz
 QUEUE:ws2
  TYPE:fcfs
  CLASS LIST:mess2 con2
    SERVICE TIMES:mess con

 QUEUE:human3
  TYPE:fcfs
  CLASS LIST:cthink3 athink3
    SERVICE TIMES:think thinkz
 QUEUE:ws3
  TYPE:fcfs
  CLASS LIST:mess3 con3
    SERVICE TIMES:mess con

 QUEUE:human4
  TYPE:fcfs
  CLASS LIST:cthink4 athink4
    SERVICE TIMES:think thinkz
 QUEUE:ws4
  TYPE:fcfs
  CLASS LIST:mess4 con4
    SERVICE TIMES:mess con

 QUEUE:human5
  TYPE:fcfs
  CLASS LIST:cthink5 athink5
    SERVICE TIMES:think thinkz
 QUEUE:ws5
  TYPE:fcfs
  CLASS LIST:mess5 con5
    SERVICE TIMES:mess con

 QUEUE:human6
  TYPE:fcfs
  CLASS LIST:cthink6 athink6
    SERVICE TIMES:think thinkz
 QUEUE:ws6
  TYPE:fcfs
  CLASS LIST:mess6 con6
```

```
      SERVICE TIMES:mess con

  QUEUE:human7
    TYPE:fcfs
    CLASS LIST:cthink7 athink7
      SERVICE TIMES:think thinkz
  QUEUE:ws7
    TYPE:fcfs
    CLASS LIST:mess7 con7
      SERVICE TIMES:mess con
.
.
.
.
.
.


  QUEUE:human30
    TYPE:fcfs
    CLASS LIST:cthink30 athink30
      SERVICE TIMES:think thinkz
  QUEUE:ws30
    TYPE:fcfs
    CLASS LIST:mess30 con30
      SERVICE TIMES:mess con

  QUEUE:human31
    TYPE:fcfs
    CLASS LIST:cthink31 athink31
      SERVICE TIMES:think thinkz
  QUEUE:ws31
    TYPE:fcfs
    CLASS LIST:mess31 con31
      SERVICE TIMES:mess con

  QUEUE:human32
    TYPE:fcfs
    CLASS LIST:cthink32 athink32
      SERVICE TIMES:think thinkz
  QUEUE:ws32
    TYPE:fcfs
    CLASS LIST:mess32 con32
      SERVICE TIMES:mess con

SET NODES:set1
  ASSIGNMENT LIST:jv(1) = 1
SET NODES:set2
  ASSIGNMENT LIST:jv(1) = 2
SET NODES:set3
  ASSIGNMENT LIST:jv(1) = 3
```

```
SET NODES:set4
  ASSIGNMENT LIST:jv(1) = 4
SET NODES:set5
  ASSIGNMENT LIST:jv(1) = 5
SET NODES:set6
  ASSIGNMENT LIST:jv(1) = 6
SET NODES:set7
  ASSIGNMENT LIST:jv(1) = 7
SET NODES:set8
  ASSIGNMENT LIST:jv(1) = 8
SET NODES:set9
  ASSIGNMENT LIST:jv(1) = 9
SET NODES:set10
  ASSIGNMENT LIST:jv(1) = 10
SET NODES:set11
  ASSIGNMENT LIST:jv(1) = 11
SET NODES:set12
  ASSIGNMENT LIST:jv(1) = 12
SET NODES:set13
  ASSIGNMENT LIST:jv(1) = 13
SET NODES:set14
  ASSIGNMENT LIST:jv(1) = 14
SET NODES:set15
  ASSIGNMENT LIST:jv(1) = 15
SET NODES:set16
  ASSIGNMENT LIST:jv(1) = 16
SET NODES:set17
  ASSIGNMENT LIST:jv(1) = 17
SET NODES:set18
  ASSIGNMENT LIST:jv(1) = 18
SET NODES:set19
  ASSIGNMENT LIST:jv(1) = 19
SET NODES:set20
  ASSIGNMENT LIST:jv(1) = 20
SET NODES:set21
  ASSIGNMENT LIST:jv(1) = 21
SET NODES:set22
  ASSIGNMENT LIST:jv(1) = 22
SET NODES:set23
  ASSIGNMENT LIST:jv(1) = 23
SET NODES:set24
  ASSIGNMENT LIST:jv(1) = 24
SET NODES:set25
  ASSIGNMENT LIST:jv(1) = 25
SET NODES:set26
  ASSIGNMENT LIST:jv(1) = 26
SET NODES:set27
  ASSIGNMENT LIST:jv(1) = 27
SET NODES:set28
  ASSIGNMENT LIST:jv(1) = 28
SET NODES:set29
```

```
  ASSIGNMENT LIST:jv(1) = 29
SET NODES:set30
  ASSIGNMENT LIST:jv(1) = 30
SET NODES:set31
  ASSIGNMENT LIST:jv(1) = 31
SET NODES:set32
  ASSIGNMENT LIST:jv(1) = 32
SET NODES:setx
  ASSIGNMENT LIST:jv(2) = 1
DUMMY NODES:dum1 dum2 dum3
CHAIN:int
  TYPE:open
  SOURCE LIST:s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15
  ARRIVAL TIMES:arv
  SOURCE LIST:s16 s17 s18 s19 s20 s21 s22 s23 s24 s25 s26 s27
  ARRIVAL TIMES:arv
  SOURCE LIST:s28 s29 s30 s31 s32
  ARRIVAL TIMES:arv

  :s1->set1->cthink1->con1->messx
  :athink1->mess1
  :s2->set2->cthink2->con2->messx
  :athink2->mess2
  :s3->set3->cthink3->con3->messx
  :athink3->mess3
  :s4->set4->cthink4->con4->messx
  :athink4->mess4
  :s5->set5->cthink5->con5->messx
  :athink5->mess5
  :s6->set6->cthink6->con6->messx
  :athink6->mess6
  :s7->set7->cthink7->con7->messx
  :athink7->mess7
  :s8->set8->cthink8->con8->messx
  :athink8->mess8
  :s9->set9->cthink9->con9->messx
  :athink9->mess9
  :s10->set10->cthink10->con10->messx
  :athink10->mess10
  :s11->set11->cthink11->con11->messx
  :athink11->mess11
  :s12->set12->cthink12->con12->messx
  :athink12->mess12
  :s13->set13->cthink13->con13->messx
  :athink13->mess13
  :s14->set14->cthink14->con14->messx
  :athink14->mess14
  :s15->set15->cthink15->con15->messx
  :athink15->mess15
  :s16->set16->cthink16->con16->messx
  :athink16->mess16
```

```
:s17->set17->cthink17->con17->messx
:athink17->mess17
:s18->set18->cthink18->con18->messx
:athink18->mess18
:s19->set19->cthink19->con19->messx
:athink19->mess19
:s20->set20->cthink20->con20->messx
:athink20->mess20
:s21->set21->cthink21->con21->messx
:athink21->mess21
:s22->set22->cthink22->con22->messx
:athink22->mess22
:s23->set23->cthink23->con23->messx
:athink23->mess23
:s24->set24->cthink24->con24->messx
:athink24->mess24
:s25->set25->cthink25->con25->messx
:athink25->mess25
:s26->set26->cthink26->con26->messx
:athink26->mess26
:s27->set27->cthink27->con27->messx
:athink27->mess27
:s28->set28->cthink28->con28->messx
:athink28->mess28
:s29->set29->cthink29->con29->messx
:athink29->mess29
:s30->set30->cthink30->con30->messx
:athink30->mess30
:s31->set31->cthink31->con31->messx
:athink31->mess31
:s32->set32->cthink32->con32->messx
:athink32->mess32

:messx->snet->dum3
:mess1 mess2 mess3 mess4 mess5 mess6->cnet
:mess7 mess8 mess9 mess10->cnet
:mess11 mess12 mess13 mess14 mess15 mess16->cnet
:mess17 mess18 mess19 mess20 mess21 mess22->cnet
:mess23 mess24 mess25 mess26 mess27 mess28->cnet
:mess29 mess30 mess31 mess32->cnet

:cnet->dum2;if(jv(2) = 0)
:cnet->dum2 dum1;0.6 0.4

:dum1->cser1;if((.25*tq(ser1)<0.5*tq(ser2))and( + +
        0.25*tq(ser1)<tq(ser3)))
:dum1->cser2;if((0.5*tq(ser2)<0.25*tq(ser1))and + +
        (0.5*tq(ser2)<tq(ser3)))
:dum1->cser3;if((tq(ser3)<0.25*tq(ser1))and + +
        (tq(ser3)<0.5*tq(ser2)))
:dum1->cser1 cser2 cser3;4/7 2/7 1/7
```

```
    :dum2->cser4;if(0.2*tq(ser4)<tq(ser5))
    :dum2->cser5;if(tq(ser5)<0.2*tq(ser4))
    :dum2->cser4 cser5;5/6 1/6
 /*
    :dum1->cser1 cser2 cser3;1/3 1/3 1/3
    :dum2->cser4 cser5;.5 .5
 */
    :cser1 cser2     cser4 cser5->setx
    :cser3->sink
 /*  :setx->dum3 sink cnet;0.10 0.7 0.20  */
    :setx->sink dnet;0.7 0.3
    :dnet->dum3
    :dum3->athink1;if(jv(1) = 1)
    :dum3->athink2;if(jv(1) = 2)
    :dum3->athink3;if(jv(1) = 3)
    :dum3->athink4;if(jv(1) = 4)
    :dum3->athink5;if(jv(1) = 5)
    :dum3->athink6;if(jv(1) = 6)
    :dum3->athink7;if(jv(1) = 7)
    :dum3->athink8;if(jv(1) = 8)
    :dum3->athink9;if(jv(1) = 9)
    :dum3->athink10;if(jv(1) = 10)
    :dum3->athink11;if(jv(1) = 11)
    :dum3->athink12;if(jv(1) = 12)
    :dum3->athink13;if(jv(1) = 13)
    :dum3->athink14;if(jv(1) = 14)
    :dum3->athink15;if(jv(1) = 15)
    :dum3->athink16;if(jv(1) = 16)
    :dum3->athink17;if(jv(1) = 17)
    :dum3->athink18;if(jv(1) = 18)
    :dum3->athink19;if(jv(1) = 19)
    :dum3->athink20;if(jv(1) = 20)
    :dum3->athink21;if(jv(1) = 21)
    :dum3->athink22;if(jv(1) = 22)
    :dum3->athink23;if(jv(1) = 23)
    :dum3->athink24;if(jv(1) = 24)
    :dum3->athink25;if(jv(1) = 25)
    :dum3->athink26;if(jv(1) = 26)
    :dum3->athink27;if(jv(1) = 27)
    :dum3->athink28;if(jv(1) = 28)
    :dum3->athink29;if(jv(1) = 29)
    :dum3->athink30;if(jv(1) = 30)
    :dum3->athink31;if(jv(1) = 31)
    :dum3->athink32;if(jv(1) = 32)


 TRACE:
END
```

7.0 References

[An85] An, J. M., and W. W. Chu, "A Resilient Commit Protocol for Real Time Systems," Proc. 1985 Real Time Systems Symposium, Dec. 1985.

[Eag86] Eager, D., E. Lazowska, and J. Zahorjan, "A Comparison of Receiver Initiated and Sender Initiated Adaptive Load Sharing," Performance Evaluation, Vol. 6 No 1, March 1986, pp. 53-68

[Arv89] Arvitzer, A., B. A. N. Ribeiro, J. W. Carlyle and W. J. Karplus, "The Advantage of Dynamic Tuning in Distributed Asymmetric Systems," UCLA-CS internal report, September 1989.

[Bog80] Boggs, D., "Internet Broadcasting," Xerox PARC technical report CSL-83-3, Palo Alto, CA, 1983.

[Chu80] Chu, W. W., et al., "Task Allocation in Distributed Data Processing," Computer, Vol. 13, No. 11, Nov. 1980, pp. 57-69.

[dSe84] E. deSouza e Silva and M. Gerla, "Load Balancing in Distributed Systems with Multiple Classes and Site Constraints," in E. Gelenbe, editor, Performance 84, Amsterdam, North Holland, 1984, pp. 17-33.

[DeV87] DeVries, J., "NFS - An Approach to Distributed File Systems in Heterogeneous Networks," Digest of Papers - Eight IEEE Symposium on Mass Storage Systems, Tucson, May 1987, pp. 77-80.

[Eth82] The Ethernet, A Local Area Network, Ver. 2 published by Digital Equipment Corporation, Intel and Xerox (the "Blue Book").

[Gai87] Gait, J., "A Distributed Process Manager for an Engineering Network Computer," Computer Research Laboratory, Tektronix, Beaverton, Oregon 97077, 1987.

[Kah88] Kahn, R. E. and V. G. Cerf, The Digital Library Project, Vol. 1: The World of Knowbots; An Open Architecture for a Digital Library System and a Plan for its Development, NRI, March 1988.

[Kle76] Kleinrock, L., Queuing Systems Vol. II, John Wiley and Sons, 1976.

[Lis83] Liskov, B., M. Herlihy, "Issues in Process and Communication Structure for Distributed Programs," Proc. 3rd Symposium on Reliability in Distributed Software and Database Systems, IEEE, 1983.

[Lis85] Liskov, B., Overview of the Argus Language and System, in Distributed Systems Methods and Tools for Specification, Paul M., H. Siegert (eds.), Berlin Heidelberg New York, 1985, pp. 343-430.

[Met76] Metcalfe, R. and D. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," Com of the ACM, 19(1976) 7, pp. 395-404.

[Mor86] Morris, J. H., M. Satyaranayanan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal, F. D. Smith, "Andrew: A Distributed Personal Computing Environment," Comm. of the ACM Vol. 29, No. 3, March 1986, pp.184-201.

[Nel84] Nelson, D., P. Leach, "The Architecture and Applications of the Apollo Domain," IEEE Computer Graphics and Applications, April 84, pp. 58-66.

[Opp83] Oppen, D. C. and Y.K. Dalal, "The Clearinghouse: A Decentralized Agent for Locating Name Objects in a Distributed Environment," ACM Transactions on Office Information Systems, Vol 1, No. 3, July 1983, pp. 230-253.

[Orr88] Orr, J.E., "Transparency, Representation and Embodied Knowledge," Presented at the 87th Annual Meeting of the American Anthropological Association, Pheonix, Arizona, Nov.19, 1988 in a panel entitled "Embodied Knowledge".

[Pet88] Peterson, L. L., "The Profile Naming Service," ACM Transaction on Computer Systems, Vol. 6, No. 4, Nov. 1988, pp. 341-364.

[Pop81] Popek, G., et al., "LOCUS: A Network Transparent High Reliability Distributed System," Proc. of the 8th Symposium on Operating System Principles, Pacific Grove, CA 1981, pp. 169-177.

[Pop85a] Popek, G. J., and B. J. Walker, Editors, The LOCUS Distributed System Architecture, The MIT Press, Cambridge, MA, 1985.

[San86] Sandberg, R., D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, Design and Implementation of the Sun Network File System, Sun Microsystems Report, Mountain View, CA 94042, 1986.

[Sau86] Sauer, C. H., A. M. Blum, P.G. Loewner, E. A. MacNair, J. F. Kurose, The Research Queuing Package Version 2: CMS Reference Manual, Nov. 1986.

[Sch85] Schatz, B., Telesophy Project Report #1, Bell Comm Research, TM-ARH-002487, August 1985.

[Sch87a] Schatz, B., "Telesophy: A System for Manipulating the Knowledge of a Community," Proc. Globecom, Tokyo, Nov. 1987.

[Sum87] Summers, R. C., "A Resource Sharing System for Personal Computers in a LAN: Concept, Design and Experience," IEEE Transactions on Software Engineering, SE-13, No. 2, August 1987, pp. 895-904.

[Sum89] Summers, R. C., "Local Area Distributed Systems," IBM System Journal, Vol. 28, No. 2, 1989, pp. 227-240.

[Ter87] Terry, D. B, and D. C. Swinehart, "Managing Stored Voice in the Etherphone System", PARC internal report Feb. 1987.

[Wag86] Wagner, Bernhard, The Cipon System, {IBM Research Center, Zurich} 1986

[Wal83] Walker, B., et al., "The LOCUS Distributed Operating System," Proc. 9th ACM Symposium on Operating Systems Principles, Vol. 17, No.5, Oct. 1983, pp. 49-70.

[Xer01] Xerox Network System Architecture, General Information Manual, XNSG 068504, April 1985.

[Xer02] Internet Transport Protocols, Xerox System Integration Standard XSIS 028112, December 1981.

[Xer03] Filing Protocol, Xerox System Integration Standard, XNSS 108605, May 1986. Publication number 610P50679.

[Xer04] Printing Protocol, Xerox System Integration Standard, XSIS 118404, April 1984.

[Xer05] Courier: The Remote Procedure Call Protocol, Xerox System Integration Standard, XSIS 038112, December 1981.

[Zim80] Zimmerman, H. "OSI Reference Model," IEEE Trans. on Communications, COM-28 (1980) 4.