AN OBJECT-ORIENTED SIMULATION MODEL FOR COMMUNICATION NETWORK TRAFFIC AT A GENERAL MAIL FACILITY

Duane L. Setterdahl
Cap Gemini America
5 Westbrook Corporate Center, Suite 600
West Chester, Illinois 60154, U.S.A.

Wayne J. Davis
Joseph G. Macro
Department of General Engineering
University of Illinois
Urbana, Illinois 61801, U.S.A.

Edward Barkmeyer
National Institute of Standards
and Technology
Gaithersburg, Maryland 20899 U.S.A.

ABSTRACT

This paper discusses an object-oriented simulation project conducted for the United States Postal System (USPS) to test the reliability of an Ethernet LAN to support communication among various equipment contained at a General Mail Facility. The study was authorized by the National Institute of Standards and Technology to verify that the communication requirements derived from the implementation of their developed Postal Equipment Management System would not saturate the communication network. The simulated network handles over 400 transactions per second, and the USPS specified that eight hours of operation (one shift) must be simulated. The simulation was programmed in C++, and the completed study demonstrated that an Ethernet LAN can reliably support the communication requirements.

1 INTRODUCTION

For the United States Postal System (USPS), the General Mail Facility (GMF) represents the primary regional processing center for the sorting and distribution of all mail items except express mail, which is handled in a separate system. The GMF contains several basic types of mail processing equipment (MPE) whose integrated operation is depicted in Figure 1:

- The Automated Face Canceling System (AFCS) cancels the postage on each mail packet to prevent its reuse.
- The Optical Character Reader (OCR) reads the address and attempts to assign the 9-digit zip code for the mail packet. If it is successful, (which is the case for approximately 75 percent of the mail packets), then the OCR prints a bar code representation of the 9-digit zip code at the bottom of the packet. If the OCR cannot assign the proper 9-digit zip code, the OCR first prints a unique bar code identifier on the back of the envelope and then captures an image of the face of the mail packet which it then sends to the Image Processing SubSystem.
- The Image Processing SubSystem (IPSS) stores the image for the mail packet with its identification number. These images are then passed to terminals staffed by postal workers, who type in the address at the keyboard until a computer can assign the proper 9-digit zip code. The zip code is then returned to the IPSS and stored with the mail packet identifier number.
- The Outgoing SubSystem (OSS) prints the correct bar code for the 9-digit zip code on the mail packets which were not successfully processed by the OCR. The OSS first reads the identifier bar code from the back of the mail packet and then sends a request for the corresponding zip code to the IPSS. The IPSS then responds with the zip code for the packet, and the OSS prints the appropriate bar code at the bottom of the packet.
- The Delivery Bar Code Sorter (DBCS) sorts the mail using the bar code for the complete 9-digit zip code.
- The Flat Mail Sorter (FMS) is designed to sort larger letters.
- The Small Parcel Bundle Sorter (SPBS) is designed to sort parcel post packets.
- Finally, the Real-time Performance Monitoring Subsystem (RPMS) is responsible for monitoring the status of the entire configuration of MPE.

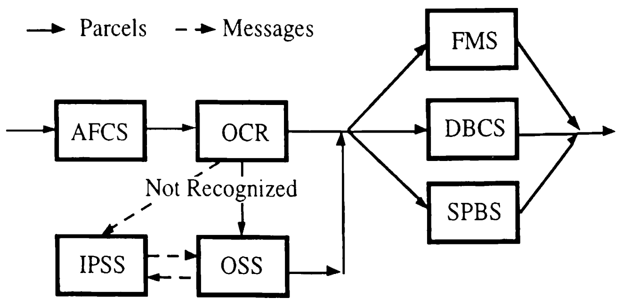

Figure 1: Integrated Operation of MPE

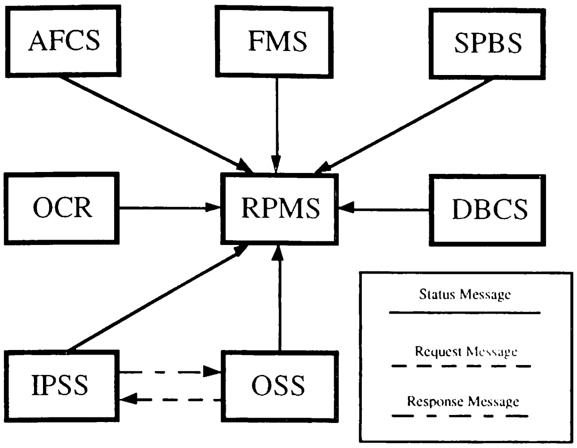

Figure 2: Communication among MPE under PEMS

All MPE is designed for high speed operation. Typically, each type of MPE, excluding the RPMS, processes 10 mail packets per second or a packet every 100 milliseconds. Operating under the Postal Equipment Management System (PEMS) developed by the National Institute of Standards and Technology (NIST) for the USPS, it was proposed that each MPE unit update its status to the RPMS after every ten mail packets are processed or approximately once per second (Figure 2).

PEMS also specifies the protocol for the interaction of the OSS with the IPSS. The OSS generates and sends a request message to the IPSS for every processed mail packet. The IPSS must respond with the correct bar code for the requesting packet within 225 milliseconds from the time that the OSS generated its request. If the correct zip code is received within the 225 millisecond interval, then the OSS prints the bar code at the bottom of the envelope. Otherwise, the specific mail packet will be diverted and must be reprocessed by the OSS.

Using Ethernet protocols, it is assumed that every message will be acknowledged. If the acknowledgment for a message is not received on time, then the message will be resent. For all messages, the acknowledgment returned by the Ethernet software provides the verification that the message was received. As a special case, when the IPSS receives a bar code request from the OSS, it generates (in addition to the acknowledge message) a response message in the form of the correct zip code.

At the request of the USPS and NIST, a simulation model was constructed to assess the ability of an Ethernet LAN to reliably support the required communication among various components of MPE at a GMF under the operation of the proposed PEMS. The simulated configuration for the GMF included 16 Delivery Bar Code Sorters, 14 Bar Code Sorters operating as Outgoing SubSystems, 9 Automated Face Cancelling Systems, 16 Flat Mail Sorters, 1 Small Parcel Bundle Sorter, 10 Optical Character Readers, 1 Image Processing SubSystem, and 1 Real-time Performance Monitoring System. This operational configuration is typically employed during the third shift when outgoing mail is being processed.

We expected that this configuration of MPE would generate approximately 210 messages per second with each message requiring an acknowledgment. Thus, over 400 transactions would be transmitted upon the network each second. The USPS also indicated that an eight-hour (28,800 seconds) shift must be simulated. During a typical eight-hour shift, we expected that over 12,000,000 transactions would be processed by the network.

In Section 2, we briefly describe the operation of the network under the PEMS protocols. In Section 3, we will provide an overview of the developed model. In the remaining sections, we will summarize the statistical results for the simulation of the investigated GMF configuration operating over an eight-hour period.

2 MODELING THE OPERATION OF THE NETWORK

In modeling the operation of the GMF network, we assumed that four types of messages are transmitted on the network: a status message from a MPE to the RPMS, a request message from the OSS to the IPSS, a response message from the IPSS to the OSS, and an acknowledgment message for every transmitted message. In Figure 3, we depict the basic steps associated with sending any message on the Ethernet from the sender to the receiver. (The standards for the operation of the ethernet and the associated time intervals were provided by NIST to the modelers.) First, the sender application generates the message and places it on the CPU's queue for processing. The processing of the message by the CPU is depicted in Figure 4. Once the message has received the attention of the CPU, the first step is to formulate the transport packet. This task requires approximately 10 microseconds to accomplish. The second task is to formulate the data packet and place it at the Ethernet card. We modeled this duration as being normally distributed with a mean of 100 and a standard deviation of 5 microseconds. After the data packet has arrived at the Ethernet card, it waits until the network is silent. When the Ethernet LAN becomes available, the sender Ethernet card transmits the message to the receiver Ethernet card.

The duration required to send the message is assumed to be 51.2 microseconds plus 100 nanoseconds per bit times the length of the message (in bits). The constant 51.2 microseconds includes the time to transmit the header, which is assumed to be the same for every type of message. Therefore, the body of the message includes only information generated by the PEMS. Status messages have a length of 960 bytes; request and response messages are 64 bytes; and an acknowledgment contains all essential information

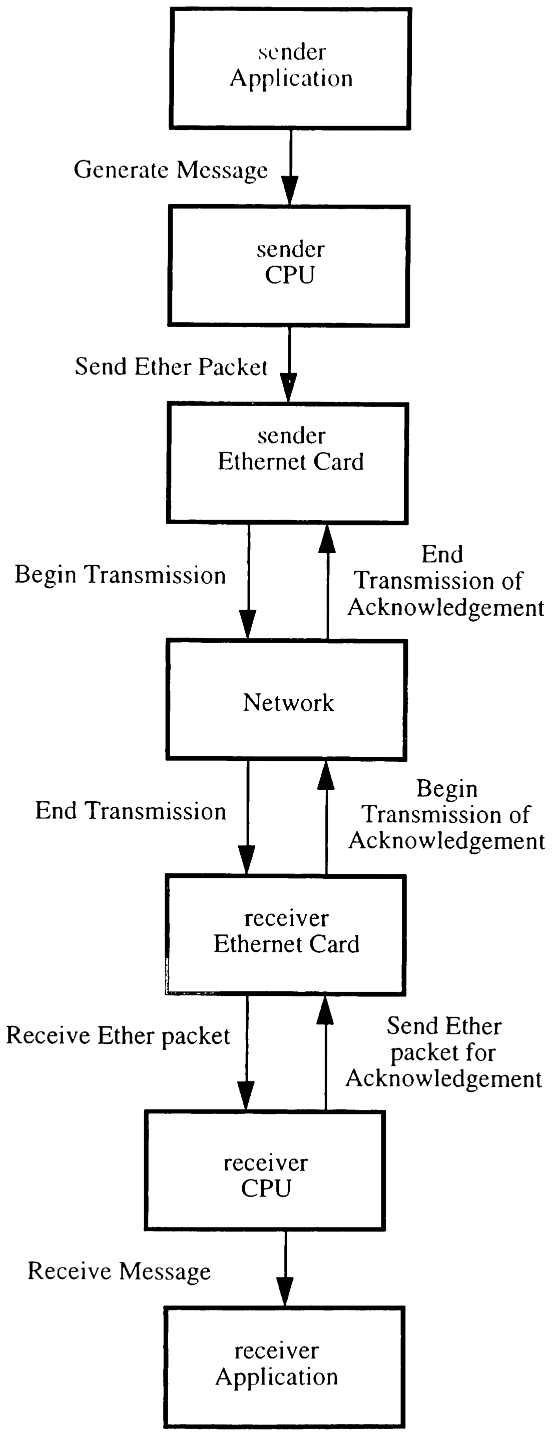

Figure 3: Steps Executed When Sending a Message

in its header and has zero length. In addition, between each sent message, there must be a silent period (or interframe gap) of 9.6 microseconds. Note that this gap time will not be counted toward the total network busy time, but it does provide an essential overhead in the operation of the network.

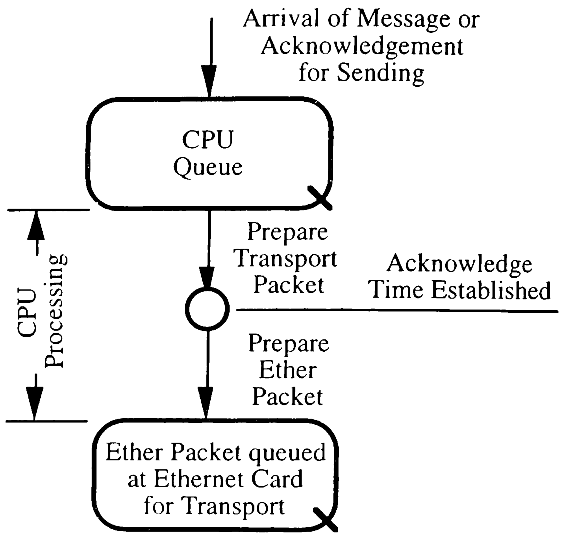
Arrival of Message or Acknowledgement for Sending
CPU Queue
CPU Processing
Prepare Transport Packet
Acknowledge Time Established
Prepare Ether Packet
Ether Packet queued at Ethernet Card for Transport

Figure 4: CPU Processing of an Outgoing Message or Acknowledgment

Two special situations are modeled here. Since there is no controller to assert which message will be transmitted next, if more than one sender is prepared to send a message when the network becomes available, all available messages are assumed to initiate transmission simultaneously. This situation leads to a network collision. If a collision occurs, then each sender will pause (or back-off) for a random delay before trying to send the message again. This delay is computed (individually by each sender) by first sampling a uniformly generated integer between 0 and 1023 and then multiplying the sampled integer by a prespecified constant, denoted as the slot time and specified here as 51.2 microseconds. Thus, a transmitter could wait for over 50 milliseconds before attempting to retransmit the message. After the network experiences a collision, it is jammed for 9.6 microseconds during which no new messages can be sent.

A second special situation occurs when the message's data is corrupted during transmission. The probability that a given bit of information will be corrupted in the transmission is approximately one in 10 million (10^7). If the message is corrupted, it is assumed to die after transmission and will not be assigned to the receiver for further processing.

After the message reaches the receiver Ethernet card, the receiver CPU is alerted and the tasks shown in Figure 5 are implemented. First, the incoming data packet is translated into a transport packet. At this point, the message acknowledgment is formulated. As illustrated in Figure 5, two tasks must be initiated at this point. However, since there is only one CPU, the tasks will be performed serially. Next, the acknowledgment is treated as a message arrival in Figure 4. Specifically, transport and data packets for the acknowledgment are formulated, and then a data packet is

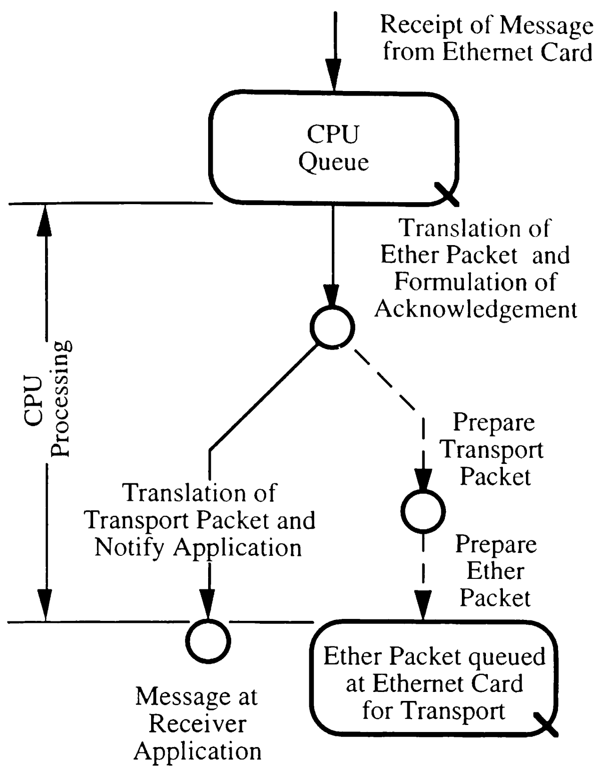
Receipt of Message from Ethernet Card
CPU Queue
Translation of Ether Packet and Formulation of Acknowledgement
CPU Processing
Prepare Transport Packet
Translation of Transport Packet and Notify Application
Prepare Ether Packet
Ether Packet queued at Ethernet Card for Transport
Message at Receiver Application

Figure 5: CPU Processing of an Incoming Message

placed upon the receiver's Ethernet card for transmission to the original sender. The durations for these tasks are assumed to be similar to those for the processing of the original message. The transmission of the acknowledgment across the LAN is also similar to that for the original message. We will discuss the handling of the acknowledgment at the original sender shortly.

Returning to the receiver CPU in Figure 5, the transport packet for the original message is now translated and forwarded to the receiver application. At this point, the transmission of the original message is assumed to be complete, and the receiver CPU can be assigned to its next task. The duration for the last translation and forwarding to the application is assumed to be normally distributed with a mean of 100 and a standard deviation of 5 microseconds.

We now discuss the handling of the acknowledgment when it arrives at the Ethernet card for the sender of the original message. In Figure 4, we note that after the transport packet for the original message was formulated, the time at which an acknowledgment must be received was established. If the transport layer (CPU) does not receive an acknowledgment of its transmission before the established deadline, the message will be re-sent. This resending process requires that the sender CPU formulate another transport and data packet and place it upon the Ethernet card as depicted in Figure 4.

The steps in Figure 3 depict the sending and receipt of a single message only. This situation is similar to a given MPE sending its status information to the RPMS. The interaction of the OSS with the IPSS is not completely detailed in Figure 1. When the OSS sends its request to the IPSS for a bar code, Figure 3 can be employed for the transmission of the request with the OSS as the sender and the IPSS as the receiver. When the message reaches the IPSS application, IPSS must formulate its response. The duration needed to formulate this response is assumed to be normally distributed with mean of 10 and standard deviation of 1 microsecond. Once the response message is formulated, the IPSS becomes the sender and the requesting OSS becomes the receiver (in Figure 1) and the entire process is repeated. Note that both the request by the OSS and the response by the IPSS will be acknowledged.

As stated above, the timely acknowledgment of a status message to the RPMS is not critical in the sense that it will not delay the operation of the sending MPE. For the OSS/IPSS interaction, the timely response by the IPSS to the OSS request is critical. At the moment in which the OSS generates its request and places it on its CPU queue, a clock is initiated. The response by the IPSS must be returned to the OSS application in a pre-specified interval of 225 milliseconds. If the response does not arrive on time, then the correct bar code cannot be printed upon the requesting packet. In this case, the mail packet must be diverted and reprocessed by the OSS. Obviously, reprocessing mail packets reduces the efficiency of the General Mail Facility and should be minimized.

3 THE MODELING APPROACH

The approach adopted for this study was influenced by several factors. First, the budget size would not permit the acquisition of a special purpose simulation tool to model communication networks. Second, there were the detailed operational characteristics which had to be addressed. NIST specified that every detail of the network's operation, (described above), had to be modeled. We initially attempted to use one of the general purpose simulation tools such as Siman (Pegden et al. 1990) or SLAM (Pritsker 1986) to model the network dynamics but soon found that this was not a feasible approach. In fact, we requested that a special version of Siman be compiled to provide a double precision representation for the simulation time, given the microsecond subintervals of time which had to be considered across an eight-hour period. Third, the eight-hour length for the simulation run was deemed essential. We, in conjunction with NIST, attempted to persuade the USPS that it was unnecessary to model eight hours of activity. That is, if the network was going to fail due to saturation, it would do so in a few short minutes. Recall that over 400 transactions were to be modeled each second. In each minute of operation, nearly 25,000 transactions would be addressed. Seldom do we address this number of simulated entities in an entire simulation run. The USPS repeated their request, and the operations of full eight-hour shift were investigated.

Given the number of message transactions that must be considered in analyzing an eight hour shift, we chose to develop the model in C++ using the object-oriented paradigm formulated by Booch (1994). We also reviewed the object-oriented modeling approaches developed by Zeigler (1990) especially for the simulation task. The algorithms used to generate random deviates were taken from Law and Kelton (1991).

Although the development of the model took only a one person-month, the effort was significant. Objects were used to model every element of the simulation including: equipment, queues, messages, and events. Given that over 14,000,000 messages transactions were processed in an eight-hour shift, literally hundreds of millions of objects were constructed and destructed during a single simulation run.

Developing the object-oriented model using C++ did permit us to model and validate every operational detail with our clients, NIST and the USPS. In fact, modeling in C++ eliminated the need to make simplifying assumptions that are usually essential to generate a model using the constructs of a given simulation language. The model that resulted was extremely efficient in that only one to two hours was required to execute a simulation run. We only report one investigated configuration of MPE here. Several others were considered as the project evolved. In fact, our developed simulation software would permit NIST to specify the number of each of the equipment type to be considered for a given simulation study. The programming was implemented in ASCII standard C++, permitting it to be executed on virtually any workstation. We executed it on both SUN and Motorola workstations to verify its portability.

In retrospect, this simulation study was a major benefit to our laboratory. It verified the reliability of the network which is reported in the next section. It also introduced us to a much more sophisticated method for developing simulation models. Using off-the-shelf packages, we were constantly attempting to program a given behavior within the constraints of the language. This was particularly apparent when we modeled flexible manufacturing systems. Earlier simulation studies conducted in our laboratory demonstrated the importance of modeling controller interactions which coordinate the flow of all entities including jobs and supporting resources, such as tools (see Dullum and Davis 1992 and Davis et al. 1993). As a result of this study, we now develop the complete object-oriented definition of the modeled system before attempting to employ a simulation language. If the requirements can be satisfied using a conventional simulation tool, then we use it. In most cases, however, we find ourselves using C++ and modifying objects which we have previously developed whenever possible.

4 RESULTS FOR THE SIMULATION STUDY

In Table 1, we summarize the configuration of MPE considered in this study. In Table 2, we summarize the statistics for the ether network. Observing that the total load of 3011 seconds for the network and that the length of the simulation run is 28,800 seconds, the network is loaded at approximately 11%, which is well below saturation. The calculated total load percentage does not include the interframe gap times or the jam times after a collision has been experienced. In retrospect, we should have added these interframe gap times and jam times to the total load time to compute the total busy time. Using the statistics on the messages sent, we can estimate the time associated with

Table 1. Simulated MPE Configuration

Number of BCS:	16
Number of OSS:	14
Number of AFCS:	9
Number of FMS:	16
Number of OCR:	10
Number of IPSS:	1
Number of RPMS:	2

Table 2. Reported Final Attribute Values for the Network

numCollisions	numBadMessages	originalLoad	totalLoad
203,579	191	2,343.07	3,010.81

numCollisions = number of collisions recorded by the network

numBadMessages = number of messages corrupted during transmission

originalLoad = total time (seconds) spent transmitting status, request and response messages excluding resends, retransmissions or acknowledgments.

totalLoad = total time (seconds) in transmitting all messages interframe gap times and jam times to be approximately 200 seconds, giving a total busy time of approximate 3300 seconds and a percentage busy of 11.1%.

In Table 3, we provide the statistics for the RPMS. Note that nearly 3 million status messages were received during the simulation. In Table 4, we report the final statistics for the IPSS in its interaction with the OSSs. Note that this summary does not include the status messages sent by the IPSS to the RPMS at a rate of one status message per ten response messages sent.

In Table 5, we summarize the statistics for the 14 included OSS's. It is important to note that no OSS ever received a late response to its request to the IPSS for a zip code. This is remarkable considering that over 4 million requests were sent by the OSSs to the IPSS for zip codes. Due to space limitations, we do not include the statistics for the 16 BCSs, 16 FMSs, the 10 OCRs and the 9 AFCSs employed in this study as they generate status messages only.

5 CONCLUSIONS

The primary concern in performing this simulation was to determine if an Ethernet LAN could adequately serve the various MPE's while operating under the proposed PEMS developed by NIST. A special concern was the ability of the IPSS to provide timely responses to the OSS's request for bar codes within the maximum permissible interval of 225 microseconds. In this simulation study, over 4 million requests were processed without a single late response. Even this statistic does not depict the true communication reliability projected by this simulation. In Figure 6, we provide a histogram for the empirical probability and cumulative density functions pertaining to the response interval, the time between the origination of the request at the OSS and the time that the response is received by the OSS from the IPSS, observed in the simulation study. Over 91% of the responses occurred within 5% of the permissible 225 microsecond interval or 11.25 microseconds. Over 95% occurred within 10% of the maximum interval. No responses required more than 50% of the maximum interval or 110 microseconds.

The data presented in this paper represent the results from a single simulation run which considered millions of transactions. Additional replications were performed, and as expected provided very similar results. (Due to their similar results and space considerations, their results have not be detailed.) In particular, there were no late responses from the IPSS to the OSS requests. We also analyzed other potential configurations of equipment, and again there were no late responses.

There are, however, additional concerns that have not been addressed that prevent us from positively stating that the proposed PEMS network configuration will be totally reliable. The first concern is that our current model does not consider any additional message traffic that may occur on the network. In particular, does the possibility exist for large messages to be transported on this network? If large messages do occur, they could temporarily congest the network and reduce the probability of the IPSS's response arriving on time. Our second concern arises in the availability of the CPU to process the message packets. Currently, no other loading upon the CPUs beyond that resulting from the implementation of PEMS is being considered.

Table 3. Reported Final Attribute Values for the RPMS

numStatusReceived	numCollisions	cpuBusyTime
1,957,191	25,703	645.867

numStatusReceived = total number of status messages received during simulation
numCollisions = number of collisions experienced in sending acknowledgment as RPMS sends no original messages
cpuBusyTime = total time (seconds) that the RPMS's CPU devoted to processing messages

Table 4. Reported Final Attribute Values for the IPSS

numRespSent	numAckn	numRequestRcv	numCollsn	numResnd	cpuBusyTime
4,189,297	4,189,297	4,189,297	252,098	146,028	2,378.17

numRespSent = total number of response messages sent during simulation
numAckn = the number of response messages that were acknowledged
numRequestReceived = the number of requests received from the OSS's during the simulation
numCollsn = number of collisions experienced by the IPSS
numResends = number of responses resent due to late acknowledgments
cpuBusyTime = total time (seconds) that the IPSS's CPU devoted to processing messages

Hence, the CPU is always available when it is needed. This certainly will not be the situation while the equipment is processing mail as each CPU will likely have other responsibilities associated with managing the operation of a given MPE.

The model has already been programmed in a manner that will permit these additional loads to be considered. However, the characteristics for these additional loads had not been quantified when this study was performed. Until additional simulation studies are conducted to consider this loading, it is difficult to fully endorse the proposed system. The developed model was submitted to the United States Postal System and the National Institute of Standards and Technology who will likely perform the future simulation studies.

Table 5. Reported Final Attribute Values for the OSS

OSS	numStatSnd	numReqSnd	numAckn	numRspOT	numCollsn	numResnd	cpuBusy
0	28,802	287,918	316,720	287,918	10,619	12,213	174.541
1	28,791	287,916	316,707	287,916	10,179	12,082	174.482
2	28,793	288,028	316,821	288,028	10,356	12,181	174.628
3	28,785	287,964	316,749	287,964	10,194	12,214	174.600
4	28,807	287,911	316,718	287,911	10,210	11,959	174.388
5	28,786	287,974	316,760	287,974	10,463	12,227	174.673
6	28,812	288,098	316,910	288,098	10,217	11,982	174.589
7	28,803	288,047	316,850	288,047	10,248	12,259	174.688
8	28,799	287,921	316,720	287,921	10,386	11,886	174.350
9	28,790	288,105	316,895	288,105	10,398	12,200	174.683
10	28,792	287,993	316,785	287,993	10,368	12,123	174.554
11	28,800	287,967	316,767	287,967	10,358	12,256	174.637
12	28,804	287,895	316,699	287,895	10,097	11,883	174.348
13	28,791	288,011	316,802	288,011	10,379	12,056	174.570

OSS = the OSS unit number from 0 to 13
numStatSnd = number of status messages sent
numReqSnd = number of request messages sent to the IPSS
numAckn = number of status and request messages acknowledged
numRspOT = number of responses to requests received on time within the required 225 microsecond interval
numCollsn = number of collisions experienced by the OSS
numResnd = number of messages resent due to a late acknowledgment
cpuBusy = total time (seconds) that the OSS's CPU devoted to processing messages

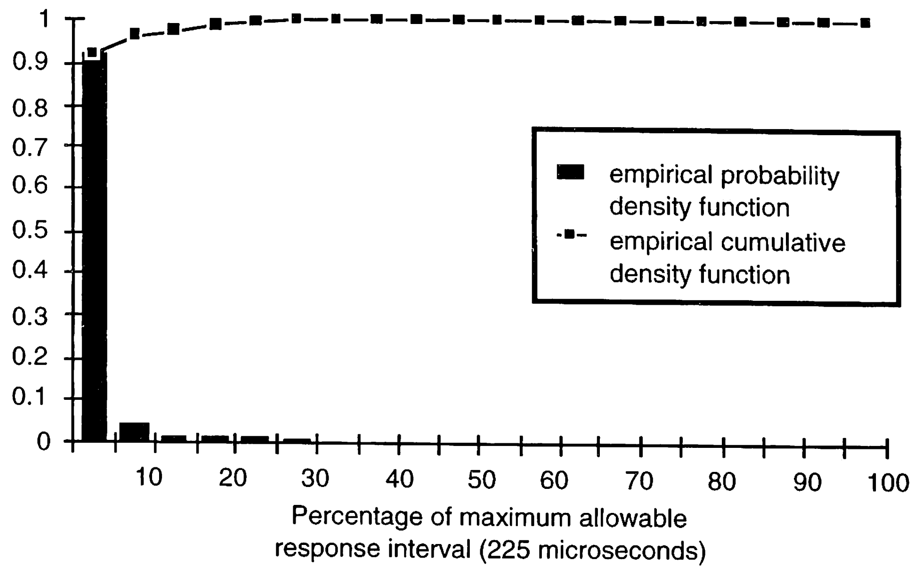

Figure 6: Histograms for Observed IPSS Response Times

REFERENCES

Booch, G. 1994. *Object-Oriented Analysis and Design with Applications.* Redwood City, California: Benjamin/Cummings Publishing.

Law, A. M., and Kelton, W. D. 1991. *Simulation Modeling and Analysis* (Second edition). New York: McGraw-Hill.

Pegden, C. D., Shannon, R. E., and Sadowski, R. P. 1990. *Introduction to Simulation Using SIMAN.* New York: McGraw-Hill.

Pritsker, A. A. B. 1986. *Introduction to Simulation and SLAM II* (Third edition). New York: Halstead-Wiley.

Zeigler, B. P. 1990. *Object-Oriented Simulation with Hierarchical Modular Models.* New York: Academic Press.

Davis, W. J., Setterdahl, D., Macro, J., Izokaitis, V., and Bauman, B. 1993. Recent Advances in the Modeling, Scheduling and Control of Flexible Automation. In *Proc. of the 1993 Winter Simulation Conference*, eds. G. W. Evans, M. Mollaghasemi, E. C. Russell, W. E. Biles, 143-155. The Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Dullum, L. M., and Davis, W. J. 1992. Expanded Simulation Studies to Evaluate Tool Delivery Systems in a FMC. *Proc. of the 1992 Winter Simulation Conference*, eds. J. J. Swain, D. Goldsman, R. C. Crain and J. R. Wilson, 978-986. The Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

AUTHOR BIOGRAPHIES

DUANE L. SETTERDAHL is systems analysis consultant for Cap Gemini America, an engineering consulting firm, in West Chester, Illinois. He received his master's degree from the Department of General Engineering at the University of Illinois.

WAYNE J. DAVIS is a professor of General Engineering at the University of Illinois. His active research areas include simulation, computer-integrated manufacturing, and real-time planning and control of discrete-event systems. He collaborates with the Automated Manufacturing Research Facility and the Electronics Manufacturing Productivity Facility. He is a member of ASME and INFORMS.

JOSEPH G. MACRO is a doctoral student in the Department of Mechanical and Industrial Engineering at the University of Illinois. He is a recipient of the US Department of Energy Integrated Manufacturing Predoctoral Fellowship.

EDWARD BARKMEYER is a systems analyst at the National Institute of Standards and Technology where he specializes in the application of computer science technologies to advanced manufacturing systems. He received his masters degree from the University of Maryland in applied mathematics.