CHAPTER 8: JACKSON NETWORKS: APPLICATIONS II

In this section we examine computer applications of Jackson queueing networks. The examples comprise

- A private branch exchange
- A multiple processor packet switching system
- An automatic call distributor
- A hierarchical analysis of a computer system
- A distributed online transaction processing system

These examples are quite lengthy, because the systems themselves are intrinsically complicated, and the requisite analysis of design tradeoffs must be done systematically. Each example involves extensive numerical studies, in order to gain insight into formulae.

8.1 Private Branch Exchange (PBX) Customer Communications Engineering

Customer premises voice switching systems *(private branch exchanges or PBXs)* can be configured with a set of trunks for connecting the local switching system to the outside telephone system. The number of trunks is typically chosen to meet a given grade of service, based on measurements of analysis (e.g., Erlang blocking formula). Currently available customer premises voice switching systems can be configured with two different types of trunks for telephone calls made outside the premises. One type is called *private line* because typically these are special or private facilities that are leased to take advantage of tariffs, and the other type is called *common carrier* because these are offered by a provider of service of the last resort (if all the private line circuits are busy, the common carrier is assumed to have circuits available for completing telephone calls). The figure below shows a hardware block diagram of such a system.

Figure 8.1.Hardware Block Diagram of PBX System

A customer hopes to take advantage of voice calling patterns, and save money, by configuring a PBX with a mix of private line and common carrier facilities. A well engineered PBX most of the time would complete calls with private line trunks, and only rarely (e.g., during a peak business hour) would there be a significant demand for common carrier trunks. Customers need not know which set of trunks were used to complete the call; only the system administrator would have this knowledge. How can this system be engineered to achieve desired performance at an acceptable price? What measurements might be gathered to quantify any statements in this realm? If call attempts can be buffered, waiting for an available trunk, how many buffers are required? Customers can be impatient, and can hang up, renege, or defect from the buffer, if the time spent waiting (in the buffer) is unacceptable. How does customer impatience impact a design? In a well engineered system, customers will only rarely defect and redial telephone calls; how can we measure how many customers are trying for the first time and how many have tried more than once to check this? These are typical of the questions we wish to address.

8.1.1 Model Figure 8.2 is a queueing network block diagram of the system to be analyzed.

Figure 8.2.Queueing Network Block Diagram of PBX

Calls arrive and are held in a buffer of maximum capacity *B* calls; if a call arrives and finds the buffer full, it is blocked and presumably will retry later.

There are *S* trunks available for calls: if less than S trunks are busy when a call arrives, it seizes a trunk for the duration of the call. If all the trunks are busy when a call arrives, the call is buffered and will either defect or renege if it does not receive some service after a given time interval, or will seize a server from a second group of *infinite* servers for the duration of a call if its waiting time exceeds a threshold.

Calls attempts arrive to a system with the interarrival times being independent identically distributed exponential random variables with mean interarrival time $1/\lambda$.

Call holding times are independent identically distributed exponential random variables with mean holding time $1/\mu$.

If a call is buffered because all trunks are busy, the sequence of time intervals until defection, given that no service was received, are assumed to be independent identically distributed exponential random variables with mean $1/\alpha$.

If a call is buffered because all trunks are busy, the sequence of time intervals until a call seizes a common carrier trunk are independent identically distributed exponential random variables with mean time out interval $1/\beta$.

If there are *B* calls in the buffer, all new arrivals are blocked or rejected. Calls are removed from the buffer in order of arrival.

8.1.2 Jackson Network Asymptotics Here are a number of tractable special cases.

First, if the buffer capacity equals the number of private line trunks $B=S$, there will be no queueing, and call attempts will be either accepted or rejected. The fraction of call attempts that are rejected is given by the Erlang blocking function:

$$fraction\ of\ blocked\ call\ attempts = B(S,\lambda/\mu) = \frac{(\lambda/\mu)^S/S!}{\sum_{K=0}^{S}(\lambda/\mu)^K/K!}$$

This provides a baseline case for comparing all subsequent studies. In a well engineered system, there should be little waiting, and little reason to have the number of buffers significantly larger than the number of private line trunks.

A second special case is to choose $B \rightarrow \infty$, which effectively means that there is *never* any buffer overflow. For this case, for the system to simply keep up with the work, we must demand

$$\lambda < S\mu + \alpha + \beta$$

In words, the arrival rate of calls must be less than the call departure rate (either due to successful completion or defection).

A final special case is to allow $\lambda \rightarrow \infty$, an overload. As $\lambda \rightarrow \infty$ the number of calls in the buffer is always B, and hence

$$\pi(B) = 1 \qquad \pi(K) = 0,\ K=0,1,...,B-1$$

EXERCISE: What happens if $S \rightarrow \infty$?

8.1.3 Summary For a private branch exchange, under normal conditions there should be no calls queueing for private line trunks: most of the calls should be completed by the private link facilities. Call buffering should occur rarely, and buffer overflow should trigger an alarm that the system is not engineered properly. The time out for overflow to common carrier facilities should be set shorter than the mean time to defect. This suggests measuring the mean number of calls completed by the private line and common carrier facilities, the number of call attempts blocked by either buffers or private line trunks not being available, the number of calls that time out, the number of calls that renege and how long they are in the system, and the time a successful call waits to be completed.

8.2 A Data Packet Switching Node

We analyze traffic handling characteristics of different configurations of a data packet switching system.

8.2.1 Functional Description The figure below shows a hardware block diagram of a multiple processor packet switching system.

Figure 8.3.Hardware Block Diagram of Packet Switching System

The companion figure below shows a functional block diagram of each step of packet handling.

Figure 8.4.Data Flow Block Diagram of Packet Switching System

Two types of transactions arrive from elsewhere in the network to be executed:

- control packets--these carry data describing a portion of the network state, maintenance information, routing information, and other network management data, but do not carry any customer data
- data packets--these carry both control information concerning the sequence number of the packet in a message and actual useful data information supplied by a customer

For control packets, there are three stages of execution:

- input handling--checking the packet to determine whether it is a control or data packet, checking the packet for errors due to transmission anomalies, and formatting it for subsequent execution
- switching and routing--updating various network management files with information on facility status at selected points in the network, modifying routing strategies to account for perceived problem areas
- output handling--formatting the control packet for subsequent transmission to another cluster in the network, adding status information concerning this cluster

For data packets, there are five stages of execution:

- input handling--checking the packet type, checking the packet for errors due to transmission anomalies, and formatting it for execution at the next stage
- switching and routing--determining the next cluster to which the packet will be sent as well as the path through the network
- output handling--formatting the data packet for subsequent transmission to another network cluster
- acknowledgement of correct receipt of the packet at the destination cluster-- the receiver of the packet now sends back a special data packet acknowledging proper receipt of the data packet which is first handled by the input handler
- switching and routing--this module updates appropriate network management files to acknowledge that the packet was correctly transmitted and then releases or discards its copy of the original data packet which was held until this point in case the transmitted packet was unsuccessfully transmitted

In any actual application, there will in fact be lost packets and a variety of failures that must be guarded against. Here we have made the assumption that everything is performing properly which hopefully would be the case in any actual system during normal operation; within this very restricted framework there are still a great many issues to be explored quantitatively.

The hardware configuration is a set of multiple processors connected by a high speed communications network or bus. The operating system allocating resources for this configuration is distributed across the processors, and administers and facilitates communication between the different modules. How many processors are needed? Should there be one processor running all this code, or more than one? Should there be functionally dedicated processors running only one portion of this code, and if so how many of each? How fast should the processors be for adequate performance? What type of delay and throughput will occur for control packets versus for data packets? How much buffering will be required in each processor? For a more detailed exposition of a data communications packet switch the interested reader is referred to the references at the end of this section.

The figure below is a block diagram of a queueing network corresponding to the above verbal description.

Figure 8.5.Queueing Network Block Diagram of Packet Switching System

Control packets migrate from cluster to cluster via steps C1, C2, C3. Data packets migrate according to steps D1, D2, D3, D4, D5, D6. Should each processor be capable of executing all steps, or should processors be dedicated to steps?

8.2.2 Traffic Handling Requirements Before determining what this system *could* do, we wish to specify what it *should* do. Our goals are as follows:

- for data packets, in the initial field system we desire a mean throughput rate of one hundred packets per second with a mean delay at a node from arrival until system buffers are flushed of all work related to that packet no larger than one hundred milliseconds during an average business hour and no larger than two hundred milliseconds during a peak business hour; two years after initial deployment we wish to handle two hundred data packets per second with the same delay during an average business hour and a peak business hour
- for control packets, for the initial field system we desire a mean throughput rate of twenty packets per second with a mean delay at a node from arrival until system buffers are flushed of all work related to that packet no larger than ten milliseconds during an average business hour and no larger than twenty milliseconds during a peak business hour; two years after initial deployment we wish to handle forty control packets per second with the same delay as the initial system

The control packet delay criterion is much more stringent than the data packet delay criterion because we wish to minimize delays in controlling the entire data communications network to prevent overload, deadlocking, and so forth. There are a variety of ways to achieve this goal and we will investigate one family of approaches in the following sections.

8.2.3 Mean Value Analysis The fraction of arriving packets that are control and data packets are denoted by F_{con}, F_{data}, respectively.

The tables below summarize the number of assembly language instructions, both application and operating system, that must be executed at each stage of execution. We have included illustrative numerical values for the symbolic parameters in order to make this more concrete.

Table 8.1.Number of Instructions Executed at Each Step per Packet

Control Packets			Data Packets		
Step Number (Fig.8.5)	*Number of Instructions of Assembly Language Symbol*	*Value*	*Step Number (Fig.8.5)*	*Number of Instructions of Assembly Language Symbol*	*Value*
C1	C_{input}	300	D1	D_{input}	300
C2	C_{switch}	1500	D2	$D_{switch}(1)$	5000
C3	C_{output}	200	D3	D_{output}	300
			D4	---	---
			D5	$D_{input}(2)$	300
			D6	$D_{switch}(2)$	750

Next, we must describe the number of processors of each type, and their respective maximum instruction rates.

Table 8.2.Processor Hardware Description

Type	*Number*	*Maximum Instruction Rate*
Input Handling	P_{input}	I_{input} instructions/sec
Switching	P_{switch}	I_{switch} instructions/sec
Output Handling	P_{output}	I_{output} instructions/sec

Our goal is to obtain an upper bound on mean throughput rate. First, we must calculate the total time required per module for each type of transaction. For brevity, we assume we have either one pool of processors doing all packet switching functions or three types of dedicated processors.

The crudest level of sizing is to assume either only control packets or only data packets. Each control packet requires 2,000 lines of assembly language code to be executed, and hence with a processor capable of executing 300,000 lines of code per second, 150 control packets per second can be executed on one processor. Each data packet requires 6,650 lines of assembly language code to be executed, and with the same speed processor 45 packets per second can be executed on one processor.

For a system with three types of processor clusters, it is useful to look at the ratio of number of lines of assembly language code executed for each of the three functions, in order to gain some feel for the approximate ratio of processor types required. Output handling requires the smallest number of lines of code per packet to be executed, and hence we normalize our results by saying that for every one output handler processor, we will require two input handler processors and from eleven processors (for a 80% control packet/20% data packet traffic mix) to eighteen processors (for a 20% control packet/80% data packet traffic mix) for routing and switching.

Table 8.3A.Single Processor Cluster--Total Time per Packet per Module

Control Packets	*Data Packets*
$\dfrac{C_{input}+C_{switch}+C_{output}}{I_{proc}\times I_{proc}}$	$\dfrac{D_{input}(1)+D_{switch}(1)+D_{output}+D_{input}(2)+D_{switch}(2)}{I_{proc}\times P_{proc}}$

Table 8.3B.Three Functionally Dedicated Processor Clusters
Total Time per Packet per Module

Processor Type	*Control Packets*	*Data Packets*
Input	$\frac{C_{input}}{I_{input} \times P_{input}}$	$\frac{D_{input}(1)+D_{input}(2)}{I_{input} \times P_{input}}$
Switching	$\frac{C_{switch}}{I_{switch} \times P_{switch}}$	$\frac{D_{switch}(1)+D_{switch}(2)}{I_{switch} \times P_{switch}}$
Output	$\frac{C_{output}}{I_{output} \times P_{output}}$	$\frac{D_{output}}{I_{output} \times P_{output}}$

We focus on two special cases:

- 20% control packets and 80% data packets--this is typical of what might be encountered in a well designed system during a peak busy hour for example
- 80% control packets and 20% data packets--this is typical of what might be encountered during severe network congestion during a peak busy hour for example

The following tables give the system capacity, in packets per second, for the case of a single cluster of processors versus three clusters of functionally dedicated processors.

Table 8.4A.Maximum Mean Throughput Rate
for Single Cluster of Processors

Number of Processors	*20% Control 80% Data*	*80% Control 20% Data*
1	52 packets/sec	102 packets/sec
2	103 packets/sec	204 packets/sec
3	156 packets/sec	306 packets/sec
4	209 packets/sec	409 packets/sec
5	261 packets/sec	511 packets/sec

Table 8.4B.Maximum Mean Throughput Rate for Three Clusters of
Functionally Dedicated Processors/One Input and One Output Processor

Number of Switching Processors	*20% Control 80% Data*	*80% Control 20% Data*
1	61 packets/sec	127 packets/sec
2	122 packets/sec	254 packets/sec
3	183 packets/sec	381 packets/sec
4	245 packets/sec	509 packets/sec
5	306 packets/sec	636 packets/sec

The capacity for the input handling processor with 20% control packets is 429 packets per second, while with 80% control packets it is 750 packets per second. The capacity for the output handling processor is independent of mix, and is 1000 packets per second. Thus, the routing and switching processor cluster is the bottleneck in the three functionally dedicated cluster implementation, and by adding processors to do that task capacity will increase linearly over the above range of values, until the input processor becomes a bottleneck.

The main thrust of traffic studies is to first make sure that the maximum mean throughput rate is well in *excess* of that required for a given application before next turning to meeting delay criteria. This excess margin of throughput will be used to assure that delay criteria are met: since the resources are not fully utilized, there will be less contention and hence shorter waiting times. Determining the maximum mean throughput rate is a necessary but by no means sufficient condition for assuring adequate system performance.

8.2.4 Delay Analysis The mean delay analysis is developed in several steps. First, if we have P identical processors each capable of a maximum instruction execution rate of I_{proc} handling one type of job with a mean of I_{job} instructions, and each processor has mean utilization U, then the mean delay per job is

$$E(T_Q) = \frac{I_{job}}{I_{proc}} \left[D(P,U) = \frac{C(P,P\ U)}{P(1-U)} + 1 \right]$$

where $C(P,P\ U)$ is the Erlang delay function discussed earlier. Second, if we allow the P identical processors to execute two different job types, with the mean number of instructions executed per job being $I_J, J=1,2$ then the mean delay for each job type is

$$E(T_{Q,J}) = \frac{U(J)}{U(1)+U(2)}\ D(P,[U(1)+U(2)]) \qquad J=1,2$$

$$U(J) = \lambda \times F_J \times \frac{I_J}{P\ I_{proc}} \qquad J=1,2$$

where F_J denotes the fraction of type J jobs that are executed over a long time interval. Finally, for the case at hand, we can calculate for each of the three workload partitions the mean delay for both control and data packets; we do so only for one case, the case of three functionally dedicated processor clusters, and omit the remaining cases for brevity:

$$E(T_{Q,control}) = \frac{C_{input}}{I_{input}} D(P_{input}, U_{input}) + \frac{C_{switch}}{I_{switch}} D(P_{switch}, U_{switch})$$
$$+ \frac{C_{output}}{I_{output}} D(P_{output}, U_{output})$$

$$E(T_{Q,data}) = \frac{D_{input}(1)+D_{input}(2)}{I_{input}} D(P_{input}, U_{input})$$
$$+ \frac{D_{switch}(1)+D_{switch}(2)}{I_{switch}} D(P_{switch}, U_{switch}) + \frac{D_{output}}{I_{output}} D(P_{output}, U_{output}) + T_{ack}$$

where T_{ack} is the acknowledgement delay.

8.2.5 Illustrative Numerical Results We now present illustrative numerical results for

- different workload partitioning
- different numbers of processors
- different mixes of control and data packets
- different processor speeds

The numbers chosen to generate the curves that follow are those described in the previous section for number of lines of assembly language code per packet.

In the study that follows, we assume that all the processors are identical, and execute either 0.3 or 0.5 million assembly language instructions per second* which is felt to be typical of current technology, based on discussions with numerous knowledgable workers in the field plus a variety of trade publications.

We vary the mix of control and data packets from 20% control and 80% data to 80% control and 20% data.

* One million instructions per second is abbreviated to one *mip* in current common usage.

The following cases are studied:

- One processor cluster executing input handling, switching, and output handling, with from one to five processors in the cluster
- Three processor clusters, one cluster for input handling, one cluster for switching, and one cluster for output handling, with from one to five switching processors and one processor handling input and one processor handling output

The remaining case, one cluster for input and output handling, and one for switching, is omitted for brevity, and its performance should be bounded by the above two extreme points.

Acknowledgement delay is varied from ten to fifty milliseconds.

We wish to calculate or approximate the largest total mean packet arrival rate such that a given set of delay criteria is met. The goals we wish to meet are the mean data packet delay, including acknowledgement, at a given node should not exceed one hundred milliseconds, while the mean control packet delay at a given node should not exceed ten milliseconds.

We first examine the data packet handling characteristics for different values of acknowledgement delay and for different mixes. We tabulate the results below:

Table 8.5A.Maximum Total Mean Arrival Rate with Mean Data Packet Delay Less Than 100 msec--20% Control/80% Data Packets--0.3 MIPS per Processor

Total Number of Processors	*Single Cluster Acknowledgement Delay 10 msec*	*Single Cluster Acknowledgement Delay 50 msec*	*Number of Switching Processors*	*Three Clusters Acknowledgement Delay 10 msec*	*Three Clusters Acknowledgement Delay 50 msec*
1	35 pkts/sec	30 pkts/sec	1	40 pkts/sec	30 pkts/sec
2	85 pkts/sec	80 pkts/sec	2	105 pkts/sec	85 pkts/sec
3	140 pkts/sec	130 pkts/sec	3	165 pkts/sec	140 pkts/sec
4	185 pkts/sec	195 pkts/sec	4	225 pkts/sec	215 pkts/sec
5	235 pkts/sec	250 pkts/sec	5	270 pkts/sec	290 pkts/sec

Table 8.5B.Maximum Total Mean Arrival Rate with Mean Data Packet Delay Less Than 100 msec--80% Control/20% Data Packets--0.3 MIPS per Processor

Number of Processors	*Single Cluster Acknowledgement Delay 10 msec*	*Single Cluster Acknowledgement Delay 50 msec*	*Number of Switching Processors*	*Three Clusters Acknowledgement Delay 10 msec*	*Three Clusters Acknowledgement Delay 50 msec*
1	75 pkts/sec	60 pkts/sec	1	100 pkts/sec	75 pkts/sec
2	180 pkts/sec	155 pkts/sec	2	225 pkts/sec	190 pkts/sec
3	280 pkts/sec	260 pkts/sec	3	360 pkts/sec	400 pkts/sec
4	350 pkts/sec	370 pkts/sec	4	430 pkts/sec	460 pkts/sec
5	450 pkts/sec	470 pkts/sec	5	550 pkts/sec	600 pkts/sec

Inspection of these two tables reveals that

- the workload mix has a definite impact on performance
- adding processors can increase traffic handling faster than one might expect naively because the processors are not completely utilized

We now turn to control packet traffic handling characteristics. The results are tabulated below:

Table 8.6A.Maximum Total Mean Arrival Rate with Mean Control Packet Delay Less Than 10 msec--0.3 MIPS per Processor

	Single Cluster		*Number of*	*Three Clusters*	
Number of	*Workload Mix*		*Switching*	*Workload Mix*	
Processors	*20% Control*	*80% Control*	*Processors*	*20% Control*	*80% Control*
1	15 pkts/sec	35 pkts/sec	1	25 pkts/sec	50 pkts/sec
2	65 pkts/sec	120 pkts/sec	2	75 pkts/sec	185 pkts/sec
3	110 pkts/sec	215 pkts/sec	3	130 pkts/sec	270 pkts/sec
4	160 pkts/sec	310 pkts/sec	4	185 pkts/sec	370 pkts/sec
5	210 pkts/sec	400 pkts/sec	5	240 pkts/sec	470 pkts/sec

Based on both these tables, we see that the control packet delay criterion and not the data packet delay criterion is limiting performance for the numbers chosen here.

How sensitive is performance if we change the delay criterion? We might design the system for an average control packet delay of ten milliseconds and a peak control packet delay of twenty milliseconds. The numbers are tabulated below:

Table 8.6B.Maximum Total Mean Arrival Rate with Mean Control Packet Delay Less Than 20 msec--0.3 MIPS per Processor

	Single Cluster		*Number of*	*Three Clusters*	
Number of	*Workload Mix*		*Switching*	*Workload Mix*	
Processors	*20% Control*	*80% Control*	*Processors*	*20% Control*	*80% Control*
1	35 pkts/sec	70 pkts/sec	1	40 pkts/sec	85 pkts/sec
2	85 pkts/sec	165 pkts/sec	2	105 pkts/sec	215 pkts/sec
3	135 pkts/sec	265 pkts/sec	3	165 pkts/sec	330 pkts/sec
4	190 pkts/sec	370 pkts/sec	4	225 pkts/sec	425 pkts/sec
5	235 pkts/sec	475 pkts/sec	5	285 pkts/sec	590 pkts/sec

System engineers can then block out total network performance knowing that data packet delays are one hundred milliseconds or less per node while control packet delays are ten milliseconds or less per node on the average, and twenty milliseconds or less per node during peak loading.

What would happen if a faster processor were available? The tables below summarize the results of one such exercise assuming the maximum instruction rate of each processor is 500,000 instructions per second:

Table 8.7A.Maximum Total Mean Arrival Rate with Mean Data Packet Delay Less Than 100 msec--20% Control/80% Data--0.5 MIPS per Processor

	Single Cluster		*Number*	*Three Clusters*	
Number of	*Acknowledgement Delay*		*of Switching*	*Acknowledgement Delay*	
Processors	*10 msec*	*50 msec*	*Processors*	*10 msec*	*50 msec*
1	75 pkts/sec	65 pkts/sec	1	65 pkts/sec	55 pkts/sec
2	150 pkts/sec	140 pkts/sec	2	160 pkts/sec	150 pkts/sec
3	250 pkts/sec	235 pkts/sec	3	280 pkts/sec	260 pkts/sec
4	320 pkts/sec	310 pkts/sec	4	380 pkts/sec	355 pkts/sec
5	420 pkts/sec	400 pkts/sec	5	480 pkts/sec	460 pkts/sec

Table 8.7B.Maximum Total Mean Arrival Rate with Mean Data Packet Delay Less Than 100 msec--80% Control/20% Data--0.5 MIPS per Processor

	Single Cluster		*Number*	*Three Clusters*	
Number of	*Acknowledgement Delay*		*of Switching*	*Acknowledgement Delay*	
Processors	*10 msec*	*50 msec*	*Processors*	*10 msec*	*50 msec*
1	150 pkts/sec	120 pkts/sec	1	175 pkts/sec	155 pkts/sec
2	300 pkts/sec	280 pkts/sec	2	390 pkts/sec	360 pkts/sec
3	480 pkts/sec	455 pkts/sec	3	600 pkts/sec	570 pkts/sec
4	650 pkts/sec	630 pkts/sec	4	800 pkts/sec	270 pkts/sec
5	820 pkts/sec	800 pkts/sec	5	1030 pkts/sec	1010 pkts/sec

Table 8.8A.Maximum Total Mean Arrival Rate with Mean Control Packet Delay Less Than 10 msec--0.5 MIPS per Processor

	Single Cluster		*Number of*	*Three Clusters*	
Number of	*Workload Mix*		*Switching*	*Workload Mix*	
Processors	*20% Control*	*80% Control*	*Processors*	*20% Control*	*80% Control*
1	50 pkts/sec	100 pkts/sec	1	60 pkts/sec	120 pkts/sec
2	125 pkts/sec	260 pkts/sec	2	150 pkts/sec	340 pkts/sec
3	210 pkts/sec	425 pkts/sec	3	250 pkts/sec	550 pkts/sec
4	300 pkts/sec	600 pkts/sec	4	350 pkts/sec	750 pkts/sec
5	400 pkts/sec	770 pkts/sec	5	450 pkts/sec	950 pkts/sec

Table 8.8B.Maximum Total Mean Arrival Rate with Mean Control Packet Delay Less Than 20 msec--0.5 MIPS per Processor

	Single Cluster		*Number*	*Three Clusters*	
Number of	*Workload Mix*		*Switching*	*Workload Mix*	
Processors	*20% Control*	*80% Control*	*Processors*	*20% Control*	*80% Control*
1	65 pkts/sec	140 pkts/sec	1	85 pkts/sec	180 pkts/sec
2	155 pkts/sec	350 pkts/sec	2	185 pkts/sec	380 pkts/sec
3	245 pkts/sec	425 pkts/sec	3	285 pkts/sec	600 pkts/sec
4	315 pkts/sec	650 pkts/sec	4	375 pkts/sec	800 pkts/sec
5	430 pkts/sec	800 pkts/sec	5	475 pkts/sec	1025 pkts/sec

The benefit in going to a faster processor is greater than just a simple speed scaling! This is because the processors are not completely utilized, and an economy of scale benefit is present.

8.2.6 Summary of Performance Analysis for Data Packet Switch We conclude with a brief summary of the relevant performance parameters for the data packet switch example discussed here.

The switching module is the main bottleneck. To alleviate this problem, designers can

- add more *dedicated* or *shared* processors, making a tradeoff between parallelism achieved by pipelining versus parallelism achieved by multitasking
- fix the number of processors and vary the workload partitioning
- chose different speed proocessors
- combinations of the above

We can achieve this with either

- three processors in a cluster handling input, routing and switching, and output, assuming all processors execute 0.3 MIPS
- five processors in three clusters, with one in one cluster for input handling, one cluster of three processors for routing and switching, and one for output, assuming all processors execute 0.3 MIPS

- two processors in one cluster, with each processor handling all three tasks, assuming all processors execute 0.5 MIPS
- four processors in three clusters, with one input handling cluster of one processor, one routing and switching cluster of two processors, and one output handling cluster of one processor, assuming all processors execute 0.5 MIPS

If we fix the total workload and the total number of processors, then the greatest gains are to be made **not** by functionally dedicating processors but rather by pooling them to execute packets, if the goal is to minimize the total mean packet delay. The refinement of having some processors for input alone and others for output alone may not be justified in terms of traffic handling capacity alone, based on the numbers at hand. Even dedicating some processors to input and output handling and others to routing and switching is not justified. However, if we wish to minimize the response time for a particular type of packet, then it may prove worthwhile to add dedicated rather than shared processors.

Varying the control and data packet mix or the processor speed can have great impact on system performance for the numbers presented here. A combination of all of the above items appears to offer the greatest potential for traffic performance improvement for the numbers presented here. The tables quantify precisely the amount of gain for each of these factors.

8.2.7 Additional Reading

[1] W.Bux, *Modelling and Analysis of Store and Forward Data Switching Centres with Finite Buffer Memory and Acknowledgement Signalling,* Elektronische Rechenanlagen, **19** (4), 173-180(1977).

8.3 Automatic Call Distributor with Customer Defection

We now return to the automatic call distributor system described earlier, in order to illustrate how to include the phenomenon of *reneging* where customers defect or hang up rather than waiting for service by an agent. Even music while you wait need not keep everyone happy! In a more serious vein, this example illustrates the *versatility* of Jackson queueing network models to handle quite complicated realistic situations, involving interactions between customers, trunks and agents. The power of the method is it makes all this look easy! A variety of models are possible, only one of which is presented here. Finally, this is a standard computer communications system that virtually everyone has encountered everyday, vitally important in daily business. Suppose you were put in charge of managing such a system: the analysis presented here would suggest where to begin to look for coming to grips!

8.3.1 Model The initial ingredients in the model are

- S servers or agents
- T trunks or links
- The arrival statistics of calls
- The service statistics of calls

The arrival statistics are assumed to be Poisson, with total offered mean arrival rate λ. The call service statistics are assumed to be adequately modeled by independent identically distributed exponential random variables with mean $1/\mu$ for the mean service time per call (this includes the time an agent and customer spend talking plus time spent by the agent in call cleanup work with no customer).

After handling this model (always do this first, simply as a check or baseline case for comparison, before doing the more complicated model), we will allow customers to defect after they have been accepted but before an agent has handled them.

8.3.2 State Space The system state space is denoted by Ω which is the set of all integer valued admissible numbers of customers in the system:

$$\Omega = \{K \mid K=0,1,...,T\}$$

Figure 8.6.Automatic Call Distributor with Customer Defection Queueing Network

8.3.3 Mean Value Analysis The mean throughput rate of completing calls is denoted by λ. The mean number of occupied trunks is given by

$$E[min(K,T)] = \lambda T_{trunk}$$

where T_{trunk} is the mean trunk holding time for completed calls, including waiting for an agent. We will calculate this in the next section; for now, it suffices to note that if trunks are the bottleneck, then

$$\lambda = \frac{T}{T_{trunk}}$$

The mean number of busy agents is given by

$$E[min(K,S)] = \frac{\lambda}{\mu}$$

If agents are the bottleneck, then

$$\lambda = S\mu$$

Combining all this, we find

$$\lambda \leq \min\left[S\mu, \frac{T}{T_{trunk}}\right]$$

For the bottleneck to occur at either the trunks or the agents, we must demand that

$$\frac{T}{S} = \mu T_{trunk} \quad trunks/agent$$

If we choose $\mu=1/30$ *seconds* while $T_{trunk} = 45$ *seconds*, then we need $45/30=1.5$ trunks per agent.

8.3.4 Jackson Network Analysis Let K denote the number of customer calls in the system. The long term time averaged fraction of time that K calls are in the system is given by

$$\pi_K = \frac{1}{G} \frac{(\lambda/\mu)^K}{K!} \prod_{I=0}^{K} \max\left[1, \frac{I}{S}\right]$$

As before, the system partition function, denoted by G, will determine various measures of performance.

Many call distributors are operated with a blocking of ten to thirty per cent of offered call attempts, in an attempt to have fewer agents and equipment than would be needed for a blocking of one to five per cent. This has an impact on customer arrival statistics. Customers will defect, hang up, or abandon waiting if too much time passes before an agent handles their transaction; this occurs when more call attempts are present rather than agents. Here is one way to quantify this phenomenon, using the modeling techniques developed earlier: the call holding rate is allowed to depend upon the number of calls in the system, such that for S or fewer calls (one per agent) the call holding time is not affected at all, while for greater than S calls, the call holding rate goes up; equivalently, the mean call talking time goes down:

$$\Phi(K) = \begin{cases} K\mu & K \le S \\ S\mu + (K-S)\alpha & K > S \end{cases}$$

The new parameter α is the mean rate at which calls defect or abandon because no agent handles them. How does this impact the distribution of the number in system? We see

$$\pi_K = \frac{1}{G} \prod_{I=0}^{K} \frac{\lambda}{\Phi(K)}$$

8.3.5 Blocking We first deal with the case of no defection. The fraction of time that all T trunks are busy is given by

$$blocking = \frac{G_T}{\sum_{K=0}^{T} G_K} \quad G_K = \sum_{J=0}^{K} \prod_{I=1}^{J} \frac{\lambda/\mu}{\Phi_{agent}(J)} \quad K=0,...,T$$

$$\Phi_{agent}(J) = \begin{cases} 1 & J=0 \\ min(S,J) & J>0 \end{cases}$$

It is instructive to rewrite this in terms of Erlang's blocking function to see how the number of agents as well as the number of trunks enters into determining the blocking:

$$blocking = \frac{\rho^{T-S} B(S,a)}{1 + \rho B(S,a)[1 - \rho^{T-S}]}$$

$$B(S,A) = \frac{\frac{A^S}{S!}}{\sum_{K=0}^{S} \frac{A^K}{K!}} \quad \rho \equiv \frac{A}{S}$$

$\rho<1$ denotes the fraction of time each *agent* is busy (either talking with a customer or doing cleanup work). Since this is less than one, the blocking can be approximated (why?) by

$$blocking \approx \frac{\rho^{T-S} B(S,A)}{1 + \rho B(S,A)} \approx \rho^{T-S} B(S,A) \quad \rho<1$$

Thus the blocking can be significantly less (why?) than we might expect based on an Erlang blocking analysis alone.

Finally, how do we include defection? We simply replace G_K in the initial expression with the appropriately modified expression, and the method for calculating the blocking is still valid (although the numbers are different!) To emphasize this, we denote by $P_K, K=0,...,T$ the fraction of time there are K calls in the system.

The mean call completion rate is given by

$$\lambda(1-B) = \mu \sum_{K=1}^{T} min(K,S) P_K$$

8.3.6 Waiting Time Suppose that calls cannot defect. The fraction of time that a call waits greater than X seconds, given that it is accepted and not blocked upon arrival, is given by

$$PROB[T_W > X \mid accepted] = \frac{1}{G} \frac{(\lambda/\mu)^S/S!}{1-B} \sum_{J=0}^{T-S-1} \rho^J \sum_{K=0}^{J} e^{-S\mu X} \frac{(S\mu X)^J}{J!}$$

The mean waiting time of an accepted call is the mean number of calls divided by the mean call completion rate, given that a call is not blocked:

$$E[T_W \mid accepted] = \frac{1}{G} \frac{(\lambda/\mu)^S/S!}{\lambda(1-B)} \sum_{J=1}^{T-S-1} J \rho^J$$

Next, we allow for defection of accepted calls. The fraction of time a customer is blocked is P_T, which is the fraction of time all trunks are busy. The fraction of time that a calling customer waits without defecting and without being blocked is given by

$$PROB[T_W > X \mid no\ blocking, no\ defection] =$$

$$= \frac{1}{1-P_T} \sum_{K=S}^{T-1} P_K \left[\frac{\alpha+\mu S}{\mu S} \right]^{K-S+1} \int_X^{\infty} e^{-\tau(\alpha+\mu(K-S+1))} \frac{[\tau(K-S+1)]^{K-S+1}}{(K-S)!} d\tau$$

8.3.7 Closing Comment In a well engineered call distributor, approximating the blocking by the Erlang B analysis and analyzing delay by Erlang C analysis should be the starting point for virtually any performance analysis. This analysis is very sophisticated compared to the much more rudimentary Erlang analysis. In any event, either analysis should be confirmed by data from an actual automatic call distributor, before doing anything else. As an exercise, try and determine for what set of parameters the mean value or Erlang analysis will give misleading insight compared with the analysis developed here.

8.3.8 Additional Reading

[1] J.W.Cohen, *Basic Problems of Telephone Traffic Theory and the Influence of Repeated Calls,* Telecommunications Review, **18** (2), 49-100 (1957)

8.4 Hierarchical Analysis

Our intent in this section is to describe a hierarchical performance analysis of a model of a computer communication system. The bottommost level is processor and memory interaction. Outputs of an analysis of this subsystem will feed the next level, operating system critical region contention. Outputs of this analysis in turn feed the next level, paging for memory management to drums and file accesses to moving head disks. Finally, outputs of that analysis feed in turn the topmost level, clerks at terminals interacting with the multiple disk spindle, multiple processor computer system.

An implicit assumption in this type of analysis is that each subsystem is in a steady state. On the other hand, the time scales for each subsystem (or layer) can be and are radically different. The processor and memory contention occurs on a time scale of tens of microseconds. The operating system critical region contention occurs on a time scale of tens of milliseconds; this means that many processor and memory interactions occur relative to any operating system critical region activity. The paging memory contention and file system access activity occurs on a time scale of hundreds of milliseconds to seconds; this means that many accesses to the operating system table occur for every page or I/O related activity. Finally, human interaction and response times have a time scale of one to ten seconds, so that many page faults and file accesses occur for every human interaction.

The figure below is a block diagram of the processor memory subsystem.

Figure 8.7.Processor/Memory Hardware Block Diagram

At the hardware system level, P processors are connected to M memory subsystems via a crossbar switch, i.e., any processor can access any memory. Each processor has a local cache memory that it checks first; if the instructions or data are not in the cache, the processor accesses the appropriate memory. Execution of that job is resumed when the appropriate memory retrieval is finished. Contention arises when more than one processor demands access to the same memory subsystem.

The figure below is a block diagram of processors contending for a serially reusable operating system kernel table:

Figure 8.8.Process Contention for Operating System Table

At the operating system kernel level, only one logical abstraction of a processor (a so called *process)* at a time can access an operating system table. This is done to insure logically correct operation, e.g., the operation of table access is atomic and irreversible. Contention arises when more than one process demands access to this table.

The figure below is a block diagram of the memory paging subsystem:

Figure 8.9.Memory/Drum Subsystem Block Diagram

Each application process is segmented into *pages* of memory. At any given instant of time, only a subset of all the pages associated with a given application process need be in main memory. A fixed head disk or *drum* stores all pages that cannot fit into main memory. How large should main memory be to insure that the delay due to waiting for pages to be moved into and out of secondary storage is acceptable?

The figure below is a block diagram of operators interacting with the computer system. Operators at terminals use the system. Each operator submits the same type of job to the system, with each job consisting of a sequence of steps involving some processing time and some input/output to moving head disks. How many operators can the system support with acceptable delay with a given processor, memory, drum and moving head disk configuration?

Figure 8.10.Clerk/System Block Diagram

8.4.1 Additional Reading

[1] P.J.Kuehn *Approximate Analysis of General Queuing Networks by Decomposition*, IEEE Transactions on Communications, **27** (1), 113-126 (1979).

[2] P.J.Kuehn, *Analysis of Switching System Control Structures by Decomposition*, Archiv fuer Elektronik und Uebertragunstechnik, **34,** 52-59 (1980).

8.4.2 Processor/Memory Model Each job requires a mean total of I instructions to be completely executed by P identical processors. No job can execute in parallel with itself, i.e., at any instant of time each job is in execution on at most one processor. Each job executes on a processor with a local cache memory: the processor checks the local memory to see if the appropriate text and data are present, and if they are, executes work. If local memory does not contain the appropriate text or data, it is fetched from one of M distinct memories. Each job involves two time intervals, one for execution out of cache memory and one for waiting for memory fetches to be completed. If all the instructions and data are in cache memory, a job will execute in a mean time T_{cache}; if none of the instructions or data are in cache memory, a job will execute in a mean time T_{memory}. The fraction of time a job executes out of cache memory is denoted by Q_{cache}; the fraction of time a job executes out of main memory is denoted by $Q_{memory} = 1 - Q_{cache}$. If there are J_P active processors, where $0 \leq J_P \leq P$, then the mean job execution rate of the *system* denoted by $I(J_P)$is given by

$$I(J_P) = \frac{J_P}{Q_{cache} T_{cache} + Q_{memory} [T_{wait}(J_P) + T_{memory}]}$$

where $T_{wait}(J_P)$ is the mean waiting time for a memory box. A different way of thinking about this is that a processor executes an average of I_{cache} instructions before a memory fetch is made, and that while executing out of cache the processor completes ν instructions per second, so T_{cache} is the total mean time to execute a job, multiple time intervals each of mean duration νI_{cache}.

All processors are fed jobs from a single ready job queue, i.e., all jobs in the queue hold all requisite resources (e.g., files, memory, buffers) except for a processor. If a processor is idle, a ready job begins execution on the idle processor. Each job will make a fraction of $F_K, K=1,...,M$ accesses to memory K in its execution. Because there is contention for memory, as more and more processors are added the mean throughput rate of executing jobs will not scale linearly with the number of processors, but will increase at a slower rate. The total number of jobs, summed over all memories, equals all processors:

$$\sum_{I=1}^{M} [J_{memory,I} + J_{cache,I}] = J_P \ \ processors \qquad J_P = 1,2,...,P$$

$J_{memory,I}, J_{cache,I}$ denotes the respective numbers of jobs executing out of cache and out of main memory directly. T_{wait} and T_{memory} are related by

$$T_{wait} + T_{memory} = T_{memory} \times stretching\ factor$$

$$stretching\ factor = \frac{E_K[J_1 + \cdots + J_M]}{E_K[min(1,J_1) + \cdots + min(1,J_M)]}$$

If there is no contention for memory, the stretching factor is one (why?), while if there is memory contention, the stretching factor can be greater than one (why?).

If we refine this analysis by using a Jackson network model, then the state space is given by

$$\Omega = \{\underline{J} \mid J_C + \sum_{K=1}^{M} J_K = J_P\}$$

The fraction of time the system is in state $\underline{J}$ is given by

$$\pi_{J_P}(J_C, J_1, ..., J_M) = \frac{1}{G_{J_P}} \frac{(Q_{cache} T_{cache})^{J_{cache}}}{J_{cache}!} \prod_{K=1}^{M} \left[Q_{memory} F_K T_{memory} \right]^{J_K}$$

This can be used to explicitly calculate the stretching factor. A mean value analysis would only allow us to bound the stretching factor. A different way of thinking about this is to determine the *effective* number of processors, denoted by $\Phi(J_P)$. Even though there are P processors, the memory contention will reduce this below P. The effective number of processors is given by

$$\Phi(J_P) = \frac{G_{J_P - 1}}{G_{J_P}} T_{cache} \le \min[J_P, P]$$

This is simply the mean throughput rate multiplied by the mean time to execute a job out of cache memory.

8.4.3 Additional Reading

[1] J.Bell, D.Casaent, C.G.Bell, *An Investigation of Alternative Cache Organizations,* IEEE Transactions on Computers, **23** (4), 346-351 (1974).

[2] W.F.Freiberger, U.Grenander, P.D.Sampson, *Patterns in Program References,* IBM J.Research and Development, **19** (3), 230-243 (1975).

[3] A.J.Smith, *Multiprocessor Memory Organization and Memory Interference,* Communications of ACM, **20** (10), 754-761 (1977).

[4] A.J.Smith, *Cache Memories,* Computing Surveys, **14** (3), 473-530 (1982).

8.4.4 Operating System Table Model There are a total of J_P active processors. There are J_U processors executing unlocked reentrant code and J_L processors either waiting or executing locked nonreentrant code, with

$$J_P = J_U + J_L$$

The nonreentrant code is serially reusable. A total of I_U and I_L instructions are executed in each state, respectively. The output of the memory/processor interference analysis enters into the operating system contention at this point. The rate at which jobs execute unlocked code is given by

$$\alpha_U(J_U) = \frac{J_U}{I_U[J_U + min(1, J_P - J_U)]} \Phi[J_U + min(1, J_P - J_U)] \nu$$

The rate at which jobs execute locked code is given by

$$\alpha_L(J_L) = \frac{min[1, J_L]}{I_L(J_P - J_L + min[1, J_L])} \Phi[J_P - J_L + min(1, J_L)] \nu$$

The fraction of time the system is in state J_U, J_L is given by $\pi(J_U, J_L)$, where

$$\pi(J_U,J_L) = \frac{1}{G_{J_A}} \prod_{I=1}^{J_U} \frac{1}{\alpha_U(I)} \prod_{K=1}^{J_L} \frac{1}{\alpha_L(K)} \quad J_U+J_L=J_A$$

The effective mean number of active processors is denoted by $\Phi'(J_A)$ and is given by

$$\Phi'(J_A) = \Phi\left[E(J_U + min(1,J_A-J_U))\right]$$

This will be even lower than what we found for memory contention (why?).

8.4.5 Drum/Disk/Memory Paging Model Each job makes an average of F file system accesses, with an associated mean time per access denoted by T_{access}. A job stores its text and data in memory pages; at any instant of time, only a subset of all the memory pages associated with a job needed for execution. Those pages that are not needed can be stored on a secondary storage device, a fixed head disk called a *drum.* A page fault is said to occur when the desired next page is not in main memory but is stored on the drum and must be moved into main memory before execution can resume. The mean number of page faults per job is a function of the mean number of pages per job: the mean number of page faults per job will decrease monotonically as we increase the mean number of pages per job. We assume that we can measure the mean number of page faults per job as a function of J_A. We denote this function by $PF(J_A)$. The mean time to service a page fault is denoted by T_P.

The page fault also requires execution of additional memory management operating system code. A job requires I_E instructions to be executed, assuming *no* page fault activity, and I_{PF} instructions to be executed per page fault, so the total mean number of instructions executed per job is given by

$$mean\ number\ of\ instructions\ per\ job = I_E + I_{PF}PF(J_A)$$

The fraction of time a processor is doing useful work is the ratio of the mean number of useful nonpaging instructions executed per job I_E divided by the total mean number of instructions executed per job. The mean number of effective processors divided by the total mean number of instructions per job is the rate at which jobs are executed. The product of these two terms is the completion rate of executing jobs, denoted by $\beta(J_A)$ and is given by

$$\beta(J_A) = \frac{I_E}{I_E+I_{PF}PF(J_A)} \frac{\Phi'[J_A]}{I_E+I_{PF}PF(J_A)}$$

Finally, we need to specify the disk activity. Each job makes an average of N_K disk accesses to disk $K=1,...,D$. Each disk access requires a mean time of T_D. There is only one paging drum. The system state at any instant of time is given by $\underline{J}$ where J_P jobs are waiting or in execution at the processors, J_{PF} jobs are waiting or in page fault activity at the drum, and $J_K, K=1,...,D$ jobs are waiting or in access to disk K. The state space constraint is that the total number of active jobs J_A must equal the total number of jobs in each activity:

$$J_P + J_{PF} + \sum_{K=1}^{D} J_K = J_A$$

The fraction of time the system is in $\underline{J}$ is denoted by $\pi(\underline{J})$:

$$\pi(\underline{J}) = \frac{1}{G_{J_A}} \prod_{K=1}^{J_P} \frac{I_E \nu}{\beta(K)} (T_{PF})^{J_{PF}} \prod_{I=1}^{D}\prod_{J=1}^{J_I}(T_D N_J)^J$$

The mean throughput rate of executing jobs is given by

$$mean\ throughput\ rate = \frac{G_{J_A-1}}{G_{J_A}}$$

8.4.6 Additional Reading

[1] P.J.Denning, *Virtual Memory,* Computing Surveys, **2** (3), 153-189.

[2] P.J.Denning, G.S.Graham, *Multiprogrammed Memory Management*, Proceedings of the IEEE, **63** (6), 924-939 (1975).

8.4.7 Clerk/System Model There are C clerks at terminals. Each clerk spends a mean amount of time T_C reading and thinking and typing, and then waits for the system to respond. There are D disks. The mean processor time per job will be inflated due to execution of memory management paging code. The mean number of drum accesses per job will be inflated due to paging. Each job has a dedicated amount of memory for its own pages. The system state at any instant of time is given by J_C, the number of clerks reading and thinking and typing, J_P, the number of jobs waiting or in execution on the processors, J_{PF}, the number of jobs waiting or servicing a page fault on the drum, and $J_K, K=1,...,D$, the number of jobs waiting or servicing an access to disk K. The total number of jobs must equal the number of clerks:

$$J_P + J_{PF} + \sum_{K=1}^{D} J_K = C$$

The fraction of time the system is in state $\underline{J}$ is denoted by $\pi(\underline{J})$, and is given by

$$\pi(J_P, J_1, ..., J_D) = \frac{1}{G_C} \frac{T_C^{J_C}}{J_C!} \prod_{K=1}^{J_P} \frac{\nu/I_E}{\beta(K)} (T_{PF})^{J_{PF}} \prod_{K=1}^{D} (T_D N_K)^{J_K}$$

8.4.8 Additional Reading

[1] J.Abate, H.Dubner, S.B.Weinberg, *Queueing Analysis of the IBM 2314 Disk Storage Facility,* Journal of the ACM, **15** (4), 577-589 (1968).

[2] J.P.Buzen, *I/O Subsystem Architecture,* Proceedings of the IEEE, **63** (6), 871-879 (1975).

8.4.9 Summary In a well engineered system, there should be little contention for resources:

- Processors execute most of the time out of local cache memory, while contention for main memory should be rare
- Processors should contend rarely for a common operating system table
- Paging should occur rarely because most jobs should fit into memory
- Clerks should contend rarely for common files and processor cycles

EXERCISE: Plot the effective number of processors versus J_P, the mean number of jobs waiting or in execution for one of P processors, on the same plot, for

A. The mean value analysis $min[J_P, P]$

B. With memory contention

C. With memory contention and operating system critical region lockout

D. With memory contention, operating system critical region lockout, and secondary storage activity for paging and for file I/O

E. With memory contention, operating system critical region lockout, secondary storage activity for paging and for file I/O, and with clerks submitting jobs at random instants of time

Interpret your results.

8.4.10 Additional Reading

[1] A.A.Fredericks, *Approximations for Customer Viewed Delays in Multiprogrammed, Transaction Oriented Computer Systems,* Bell System Technical Journal, **59** (9), 1559-1574 (1980).

[2] J.C.Browne, K.M.Chandy, R.M.Brown, T.W.Keller, D.F.Towsley, C.W.Dissly, *Hierarchical Techniques for the Development of Realistic Models of Complex Computer Systems,* Proceedings of the IEEE, **63** (6), 966-975 (1975).

8.5 On Line Transaction Processing

The motivation for this section was taken from what is widely known as online transaction processing:

- Attendants or clerks receive telephone calls from customers
- Information is entered into and retrieved from an online data base by clerks talking with customers
- Workers who do not have direct customer access fill the customer order again by interacting with the online system

This particular system is involved with handling telephone repair transactions, but the principles encountered here could just as well be applied to banking, finance, distribution, transportation, or other market segments. In fact, one of the first applications of this type of system was found in the airline industry, and the hard won lessons there have been applied in numerous other applications.

The system must execute four different types of transactions:

[1] Trouble Entry (TE)--The telephone number of the problem telephone is entered into the system, and the clerk waits for the system to respond with a list of past problems associated with that telephone plus customer information plus a list of potential times and dates when a service call can be made (if necessary)

[2] Trouble Report (TR)--A more detailed description of the problem is entered into the system for use later on, or in some cases for closing out the problem

[3] Trouble Tracking (TT)--Customers and service staff interrogate the system to find the status of a particular job

[4] Testing--Actual repair staff testing, either remotely or on premises, of the facilities in question

The figures below are a representative hardware configuration and a queueing network block diagram for this system:

Figure 8.11.Hardware Configuration

Figure 8.12.Queueing Network Block Diagram

8.5.1 System Traffic Performance Goals The table below is a representative set of mean value arrival rates and mean response time goals for each type of transaction:

Table 8.9.Traffic Goals

Transaction Type	*Interarrival Frequency*	*Response Time*
Trouble Entry	1 per clerk per two minutes	5 seconds
Trouble Report	1 per clerk per two minutes	10 seconds
Trouble Tracking	1 per clerk per minute	15 seconds
Testing	1 per clerk per five minutes	45 seconds

These goals are intended for a peak busy hour: if they are met, performance is adequate. As a refinement, we might wish to specify these goals at two different points, say during a normal business day (10-11 AM or 2-3 PM), as well as during a peak busy hour (busiest day in the month, quarter, year), in order to assess sensitivity to fluctuations in offered load.

The figure below plots the total mean time to complete one customer contact (both a trouble entry and trouble report) transaction as a function of the mean number of lines that each attendant must handle. We have assumed one order per line per year, and two hundred fifty business days per year. We might wish to assign staff to this type of job for only part of a day, say two or four hours, rather than for eight hours a day, and have plotted the impact on system performance in this figure for these three choices.

Figure 8.13.Mean Throughput Rate vs Number of Lines/Clerk

We complement this plot with a second plot showing mean customer delay vs number of lines per clerk, shown below.

Figure 8.14.Mean Customer Delay vs Number of Lines/Clerk

8.5.2 Resources Required Per Transaction What are the resources required for each step of each transaction? First, we show the workflow and resource used at each step for each transaction type in the figures below:

Figure 8.15.Trouble Entry Work Flow

Figure 8.16.Trouble Report Work Flow

Figure 8.17.Trouble Tracking Work Flow

Figure 8.18.Testing Work Flow

The tables below summarize the mean time required to hold these resources at each step of execution of each transaction:

Table 8.10.Trouble Entry Work Flow Resources

Task	*Mean Time*
Transmit 30 Bytes Data from Terminal to XFE	80 msec
XFE Switching	250 msec
Transmit Data from XFE to FE	80 msec
Front End Processor	500 msec
Front End Disk Access(5 Accesses)	125 msec
Transmit Mask(800 Bytes) from FE to XFE	250 msec
XFE Switching	250 msec
XFE to Terminal Transmission(800 Bytes)	250 msec
Total	2.785 sec

Table 8.11.Trouble Report Work Flow Resources

Task	*Mean Time*
Data Transmission from Terminal to XFE(500 Bytes)	1.25 sec
XFE Switching	250 msec
XFE to FE Data Transmission	1.25 sec
Front End Processor Execution	750 msec
Front End Disk Access Time(12-15 Accesses)	300-375 msec
FE to XFE Data Transmission	125 msec
XFE Switching	250 msec
XFE to Terminal Data Transmission(50 Bytes)	125 msec
Total	4.3-4.375 sec

Table 8.12.Tester Work Flow Resources

Task	*Mean Time*
Data Transmission from Terminal to FE(150 Bytes)	750 msec
FE Processor Time	100 msec
FE Disk Access Time(2 Disk Accesses)	100 msec
Data Transmission from FE to MLT(150 bytes)	750 msec
MLT Execution	20-30 sec
MLT to FE Data Transmission(300 Bytes)	1.5 sec
FE Processor Time	100 msec
FE Disk Access Time(2-3 Disk Accesses)	100-150 msec
Print Output(900 characters, 30 char/sec)	30 sec
Total	53.4-63.45 sec

Table 8.13.Trouble Tracking Work Flow Resources

Task	*Mean Time*
Terminal to FE Data Transmission(200 Bytes)	1.0 sec
FE Processor Time	500 msec
FE Disk Access Time(5 Accesses)	250 msec
Transmit Data to Terminal(200 Bytes)	1.0 sec
Total	1.75 sec

8.5.3 Delay Analysis We choose to fix the background workload and vary the number of clerks at terminals handling TE and TR transactions. The background load we fix is

- Two hundred forty trouble status transactions per hour
- Sixty tester transactions per hour per front end

The remaining parameters that can be varied are

- Number of cross front end processors

- Number of front end subsystems
- Number of disks per front end
- Number of data links between the clerks' terminals and XFE
- Number of data links between each FE and XFE
- Work balance per FE

We begin with the simplest case, doing a single type of transaction, TE. If performance is inadequate here, it will presumably be worse with multiple transactions being executed. This also gives us a natural starting point for more detailed studies, a baseline comparison case. The table below summarizes the mean number of trouble entry transactions per hour that can be handled with a mean response time of four seconds or less, assuming

- Six front ends
- Two cross front ends
- Six terminal to XFE data links
- Two disks per FE
- 50 msec per disk access per transaction
- 250 msec FE processor time per transaction

Table 8.14.Trouble Entry Mean Throughput Rate with Mean Response Time 4 Seconds or Less with No Other Types of Transactions Executed

XFE Time	*FE Disk Access/TE*	
per TE	*5 Disk Accesses*	*25 Disk Accesses*
0.10 sec	>29,647 TE/HR	22,235 TE/HR
0.20 sec	>29,647 TE/HR	21,176 TE/HR
0.30 sec	19,059 TE/HR	17,470 TE/HR
0.40 sec	13,764 TE/HR	12,706 TE/HR
0.50 sec	10,588 TE/HR	9,529 TE/HR
0.60 sec	8,470 TE/HR	7,941 TE/HR

Each trouble entry transaction is switched twice by the XFE, once going in, once going out, and requires 250 msec per switch, so the current system is operating at a maximum of roughly ten thousand TE transactions per hour with no other work going on.

What if we include the three other transaction types? How will this impact the system performance? The table below summarizes calculations for that exercise:

Table 8.15.Trouble Entry Mean Throughput Rate with Mean Response Time 4 Seconds or Less with Other Transactions Concurrently Executed

XFE Time	*FE disk Access/TE*	
per TE	*5 Disk Accesses*	*25 Disk Accesses*
0.10 sec	28,059 TE/hr	18,000 TE/hr
0.20 sec	23,294 TE/hr	16,941 TE/hr
0.30 sec	15,353 TE/hr	13,765 TE/hr
0.40 sec	11,118 TE/hr	10,059 TE/hr
0.50 sec	8,470 TE/hr	7,412 TE/hr
0.60 sec	6,882 TE/hr	5,823 TE/hr

This impact is roughly twenty per cent: the system TE handling capability drops from 10,588 or 9,529 TE/hr to 8,470 or 7,412 TE/hr

What if the XFE were replaced with a high speed bus, i.e., such as *ETHERNET* or any of a wide number of variants? This would move performance down to the regime where the XFE time per message is under ten milliseconds, increasing capacity to 18,000 TE/hr at the worst! Note that the total system capacity increases by roughly a factor of two to three if the XFE is *infinitely* fast, because the XFE is no longer the bottleneck limiting performance.

What if we vary the number of CRSAB-XFE data links? The table below summarizes the impact on performance in going from four to six links, with five disk accesses per TE:

Table 8.16.Trouble Entry Mean Throughput Rate with Mean Response Time 4 Seconds or Less 5 Disk Accesses/TE

XFE	*25 msec/disk access*	
Time/TE	*4 CRSAB-XFE Links*	*6 CRSAB-XFE Links*
0.10 sec	29,647 TE/hr	29,647 TE/hr
0.20 sec	29,647 TE/hr	29,647 TE/hr
0.30 sec	19,059 TE/hr	19,059 TE/hr
0.40 sec	13,765 TE/hr	13,765 TE/hr
0.50 sec	10,588 TE/hr	10,588 TE/hr
0.60 sec	8,470 TE/hr	8,470 TE/hr

Hence, we see that going from four to six CRSAB-XFE data links has *negligible* impact on performance at this level of analysis.

What is the impact on performance due to file system layout? One way to quantify this is to vary the disk access time from twenty five milliseconds per access to fifty milliseconds per access:

Table 8.17.Trouble Entry Mean Throughput Rate with Mean Response Time 4 Seconds or Less Four CRSAB-XFE Data Links

XFE	*5 Disk Accesses/TE*	
Time/TE	*25 msec/Disk Access*	*50 msec/Disk Access*
0.10 sec	18,529 TE/hr	18,529 TE/hr
0.20 sec	18,529 TE/hr	18,000 TE/hr
0.30 sec	14,823 TE/hr	14,823 TE/hr
0.40 sec	10,588 TE/hr	10,588 TE/hr
0.50 sec	8,470 TE/hr	8,470 TE/hr
0.60 sec	6,682 TE/hr	6,682 TE/hr

This shows that at this level of analysis, varying the file system layout has virtually no impact on performance.

What if we go from one to two XFE systems? The table below is a summary of calculations for one such exercise:

Table 8.18.Trouble Entry Mean Throughput Rate with Mean Response 4 Seconds or Less 5 Disk Access/TE--4 CRSAB-XFE Data Links

XFE	*One*	*Two*
Time/TE	*XFE*	*XFEs*
0.10 sec	18,529 TE/hr	18,529 TE/hr
0.20 sec	13,235 TE/hr	18,000 TE/hr
0.30 sec	6,882 TE/hr	14,823 TE/hr
0.40 sec	4,765 TE/hr	11,117 TE/hr
0.50 sec	3,176 TE/hr	8,470 TE/hr
0.60 sec	2,647 TE/hr	6,882 TE/hr

This table makes clear the traffic handling capability of the *system* is roughly doubled in going from one to two XFEs, because this is the bottleneck!

What if we drop the mean response time criterion from four seconds to three seconds? The table below is a summary of one such set of calculations:

Table 8.19.Trouble Entry Mean Throughput Rate vs XFE Time per TE 5 Disk Accesses/TE--Two XFEs

XFE	*Maximum Mean Response Time*	
Time/TE	*3 Seconds*	*4 Seconds*
0.10 sec	18,000 TE/hr	18,529 TE/hr
0.20 sec	17,470 TE/hr	18,000 TE/hr
0.30 sec	14,291 TE/hr	14,823 TE/hr
0.40 sec	10,059 TE/hr	11,118 TE/hr
0.50 sec	7,941 TE/hr	8,470 TE/hr
0.60 sec	5,823 TE/hr	6,682 TE/hr

This suggests that the total system traffic handling capability is sensitive to the mean response time goal, since going from three to four seconds increases the mean throughput rate by ten per cent.

Additional studies can be carried out varying

- The amount of front end processor time per TE
- The number of front end processors
- The tester workload distribution (uniformly distributed across all front ends versus all focused on one front end)

8.5.4 Buffering Analysis What type of buffering might be employed in the XFE? Most of the time a transaction demands a buffer, a buffer should be available: the fraction of time a buffer is not available should be negligible. As an exercise, show that the fraction of time no buffer is available, given B_{buffer} buffers for a P_{XFE} processor XFE configuration is given by

$$fraction\ of\ time\ no\ buffer\ available = \frac{P\ U^{B_{buffer}+1-P_{XFE}} B_{Erlang}(P,P\ U)}{P(1-U)\,(1-B_{Erlang}(P,P\ U))}$$

where $B_{Erlang}(P,A)$ is the Erlang blocking function:

$$B_{Erlang}(P,A) = \frac{A^P/P!}{\sum_{K=0}^{P} A^K/K!}$$

and U is the fraction of time each XFE processor is busy, its utilization.

The table below summarizes one set of calculations for number of buffers for either one or two XFE processors (i.e., the calculation was not sensitive to the number of XFEs):

Table 8.20.Minimum Number of Buffers

Processor	*Fraction of Time All Buffers Filled*		
Utilization	*0.001*	*0.0001*	*0.00001*
0.50	11	14	17
0.60	14	19	23
0.70	20	27	33
0.80	32	42	52
0.90	66	88	110

In practice, it is important to *dedicate* buffers to different FE processors rather than to *pool* buffers for all FE processors, because one FE will typically be congested, and will hold its buffers for much longer than all the other FE systems, effectively locking out or hogging resources.

8.5.5 Summary First we developed a systematic description of the flow of control and data throughout the system, the resources required for each step of each job, and the mean time for each job step. Second we used this to determine bottlenecks. What are the bottlenecks? The candidates are

- The XFE processor is a bottleneck
- The FE processor is a bottleneck
- The FE disk is a bottleneck
- The MLT is a bottleneck
- The data links between the attendants and XFE are a bottleneck
- The XFE to FE data links are a bottleneck

The numbers here suggested that for virtually any scenario, under light load, the attendants are the bottleneck, while under heavy load, the XFE is the bottleneck. If the XFE is replaced by a much higher speed local area network or switch, then the FE (either processor or disk) becomes the bottleneck.

Third, we carried out sensitivity studies, that showed that

- If the system executed TE transactions alone, then the current mean throughput rate is 10,000 per hour, while if a representative mix of TE, TR, Tracking and Testing transactions are executed, then this drops to 7,500 transactions per hour.
- Once the XFE-CRSAB has four or more data links, the data links are not a significant bottleneck.
- If the file system layout is changed on the FE disk, this has virtually no impact on performance.
- Doubling the number of XFEs more than doubles the number of TEs per hour with acceptable mean response time (3,000 TE/hour with one XFE, 8,000 TE/hour with two XFEs). The reason it *more* than doubles is that the XFE is *not* executing jobs at its maximum rate, plus there is an *economy of scale* in going from one to two XFEs.
- Changing the acceptable mean response time from four to three seconds had negligible impact on performance.

8.5.6 Additional Reading

[1] R.L.Martin, *System Architecture of the Automated Repair Service Bureau,* Bell System Technical Journal, **61** (6), Part 2, 1115-1130 (1982).

[2] J.P.Holtman, *The Context-Sensitive Switch of the Loop Maintenance Operations System,* Bell System Technical Journal, **61** (6), Part 2, 1197-1208 (1982).

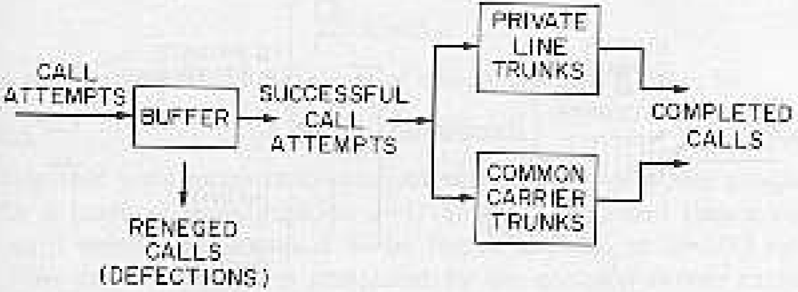


Figure 8.1.Hardware Block Diagram of PBX System

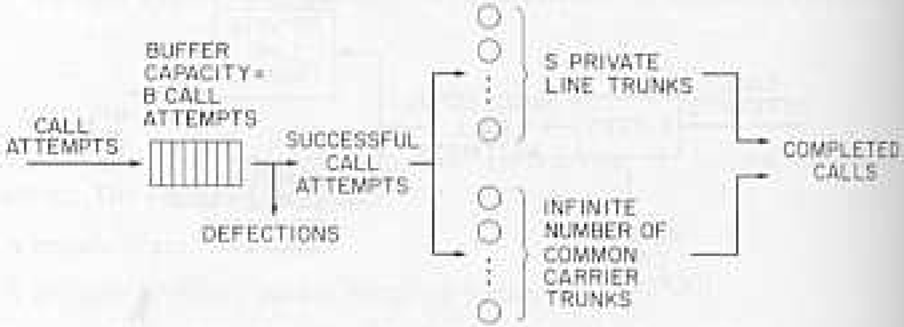


Figure 8.2.Queueing Network Block Diagram of PBX

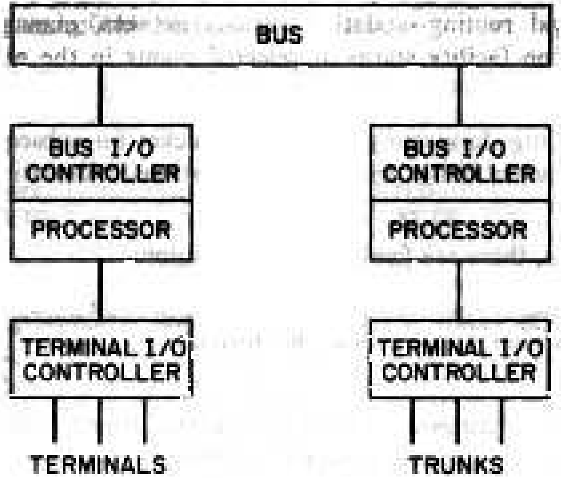


Figure **8.3.Hardware Block Diagram of Packet Switching System**

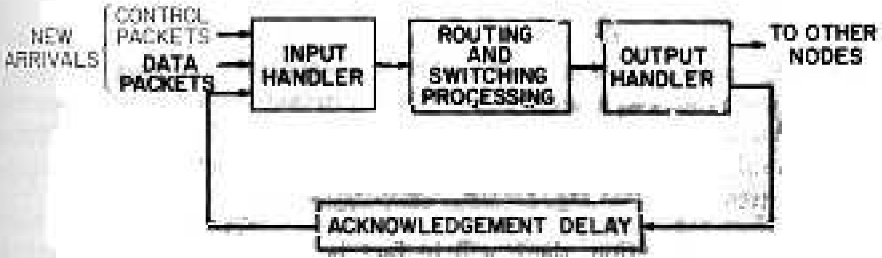


Figure 8.4.Data **Flow Block Diagram of Packet Switching System**

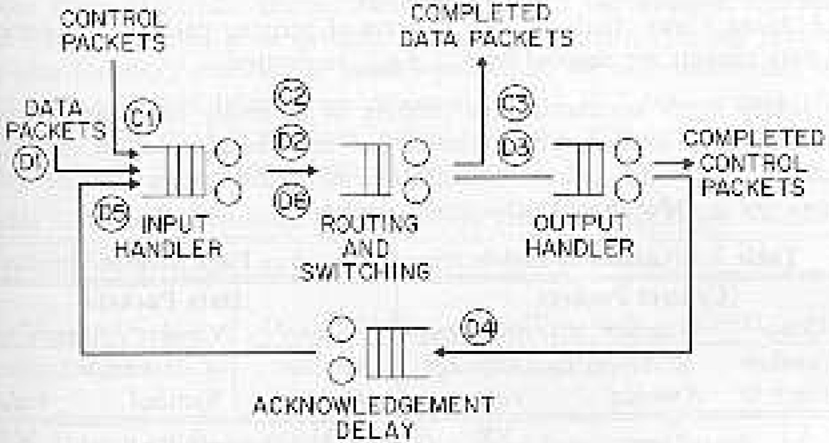
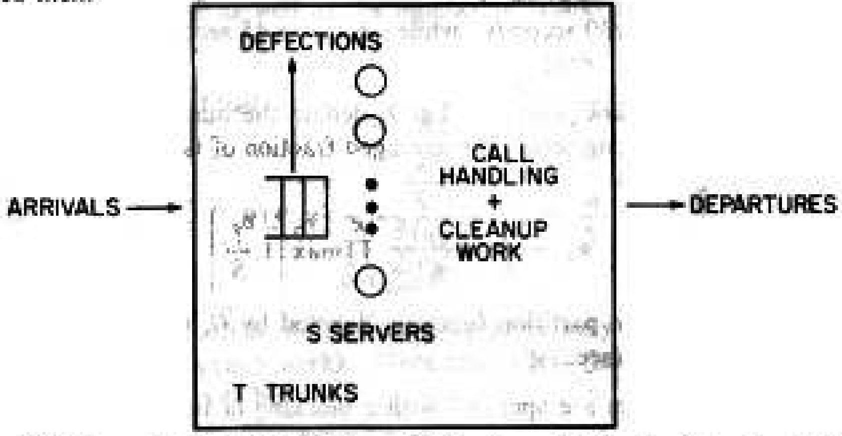


Figure 8.5.Queueing Network Block Diagram of Packet Switching System

andled them



igure 8.6.Automatic Call **Distributor with Customer Defection Queueing Network**

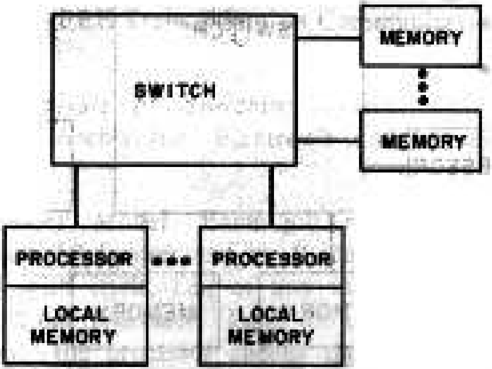


Figure 8.7.Processor/Memory Hardware Block Diagram

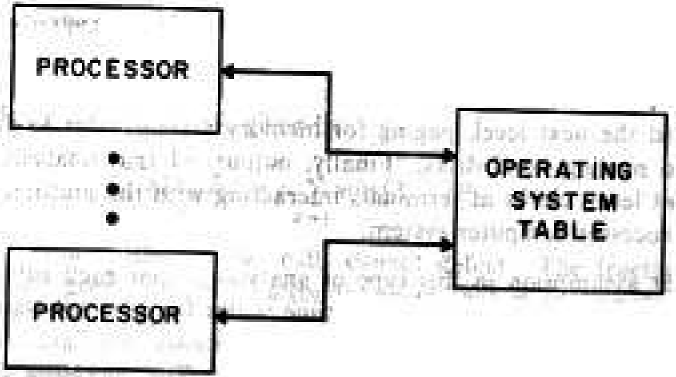


Figure 8.8.Process Contention for Operating System Table

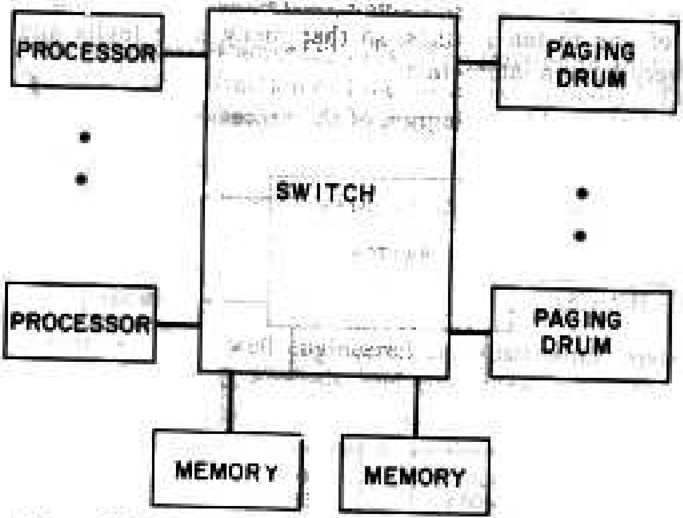


Figure 8.9.Memory/Drum Subsystem Block Diagram

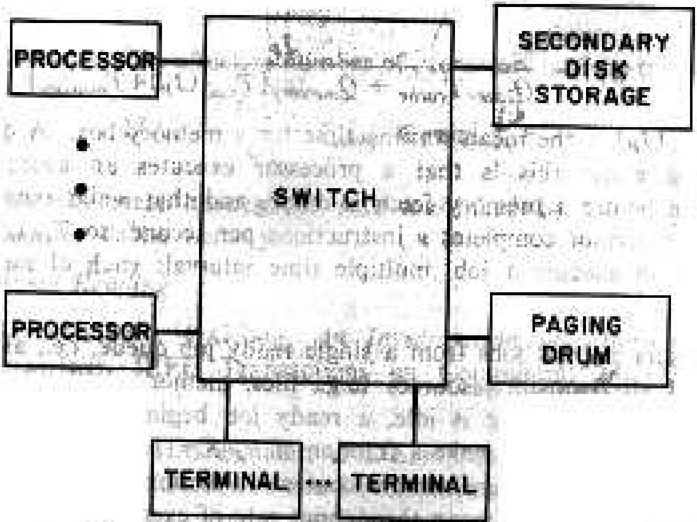


Figure 8.10.Clerk/System Block Diagram

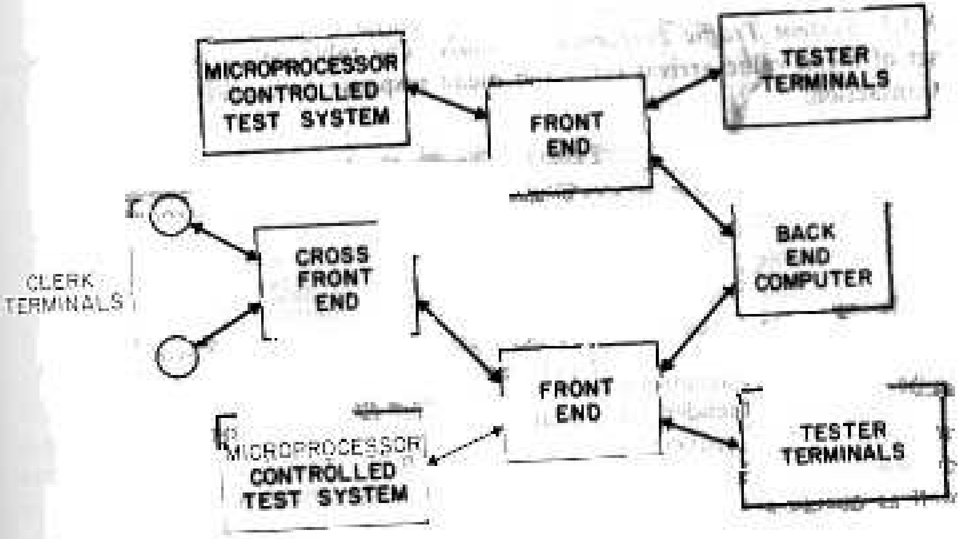


Figure 8.11.Hardware Configuration

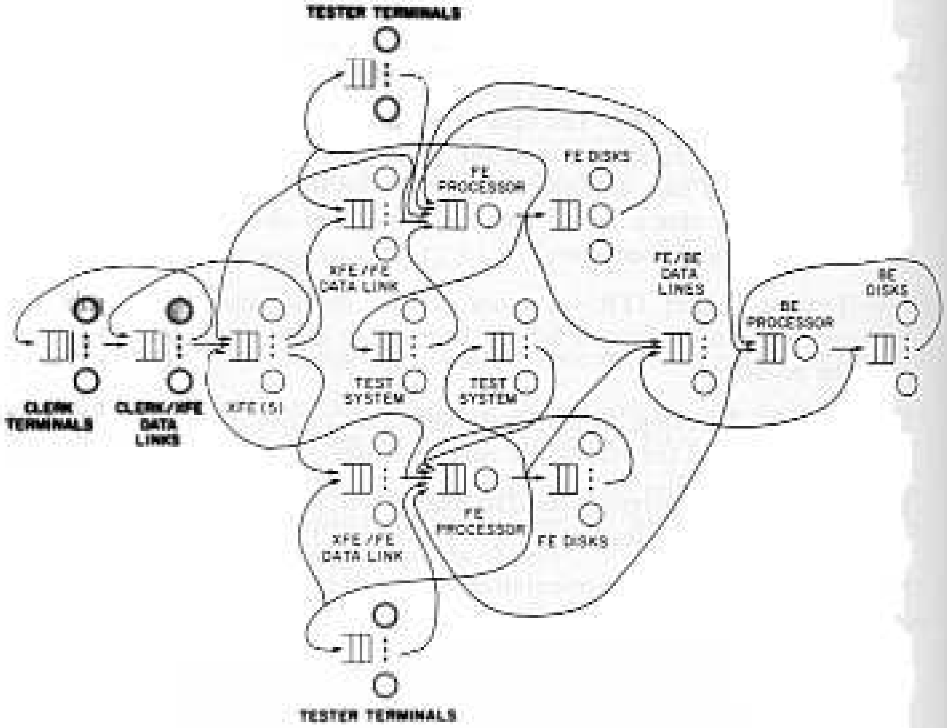


Figure 8.12.Queueing Network Block Diagram

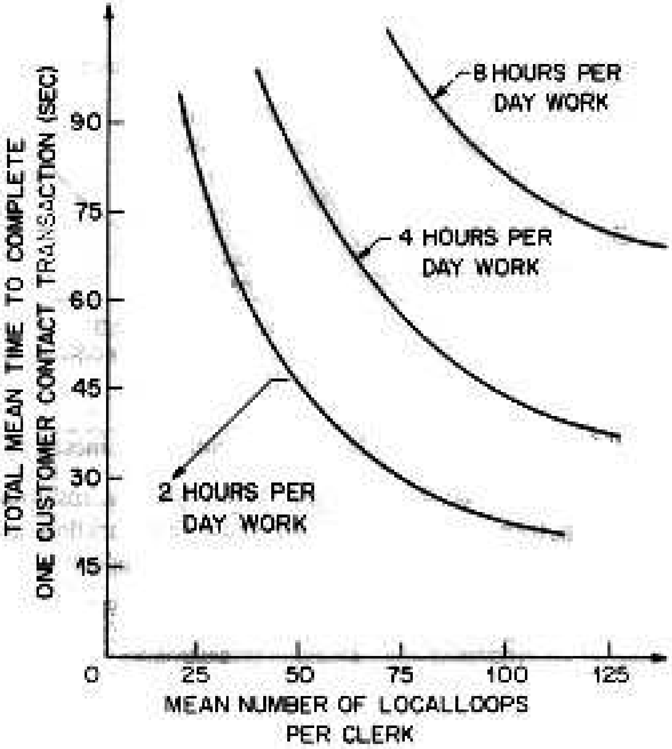


Figure 8.13.Mean Throughput Rate vs Number of Lines/Clerk

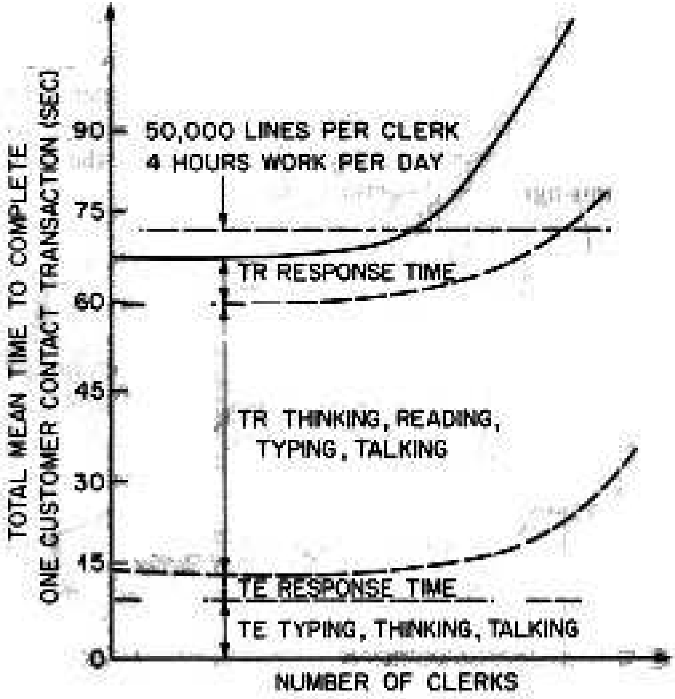


Figure 8.14.Mean Customer Delay vs Number of Lines/Clerk

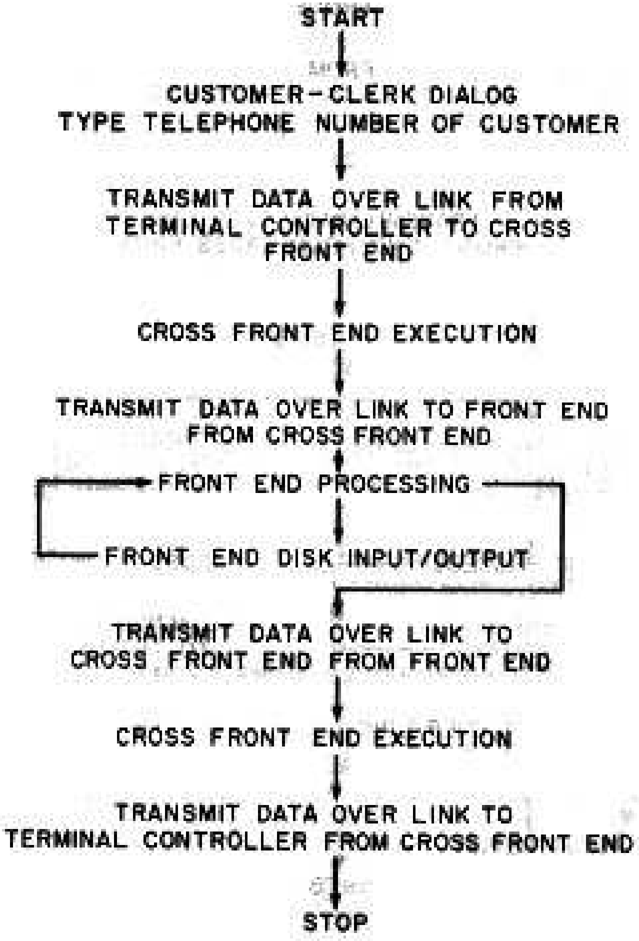

START
CUSTOMER-CLERK DIALOG
TYPE TELEPHONE NUMBER OF CUSTOMER
TRANSMIT DATA OVER LINK FROM TERMINAL CONTROLLER TO CROSS FRONT END
CROSS FRONT END EXECUTION
TRANSMIT DATA OVER LINK TO FRONT END FROM CROSS FRONT END
FRONT END PROCESSING
FRONT END DISK INPUT/OUTPUT
TRANSMIT DATA OVER LINK TO CROSS FRONT END FROM FRONT END
CROSS FRONT END EXECUTION
TRANSMIT DATA OVER LINK TO TERMINAL CONTROLLER FROM CROSS FRONT END
STOP

Figure 8.15.Trouble Entry Work Flow

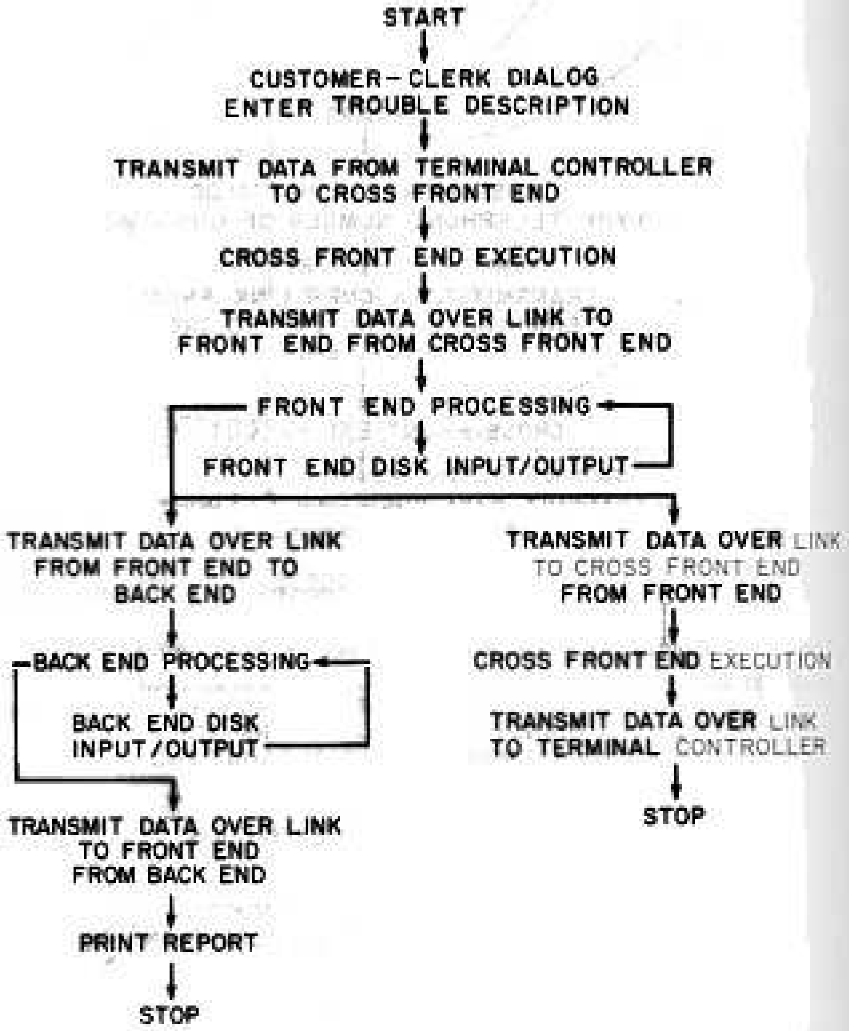

START
CUSTOMER – CLERK DIALOG
ENTER TROUBLE DESCRIPTION
TRANSMIT DATA FROM TERMINAL CONTROLLER TO CROSS FRONT END
CROSS FRONT END EXECUTION
TRANSMIT DATA OVER LINK TO FRONT END FROM CROSS FRONT END
FRONT END PROCESSING
FRONT END DISK INPUT/OUTPUT
TRANSMIT DATA OVER LINK FROM FRONT END TO BACK END
BACK END PROCESSING
BACK END DISK INPUT/OUTPUT
TRANSMIT DATA OVER LINK TO FRONT END FROM BACK END
PRINT REPORT
STOP
TRANSMIT DATA OVER LINK TO CROSS FRONT END FROM FRONT END
CROSS FRONT END EXECUTION
TRANSMIT DATA OVER LINK TO TERMINAL CONTROLLER
STOP

Figure 8.16.Trouble Report Work Flow

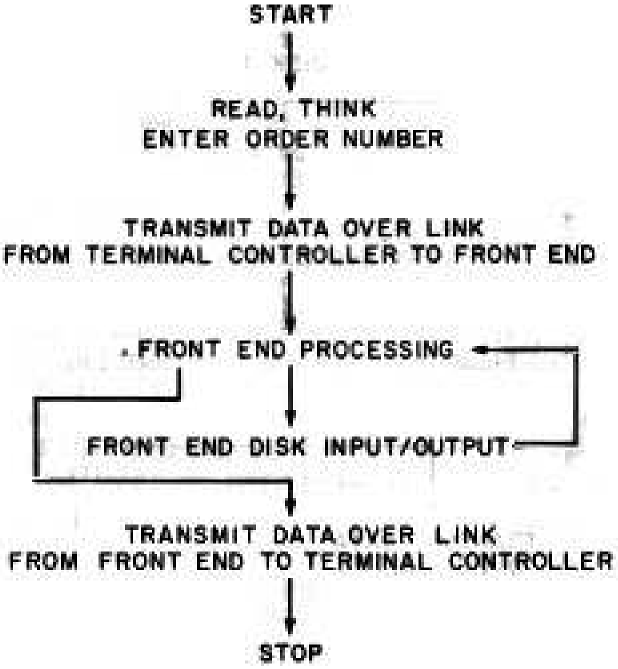


Figure 8.17.Trouble Tracking Work Flow

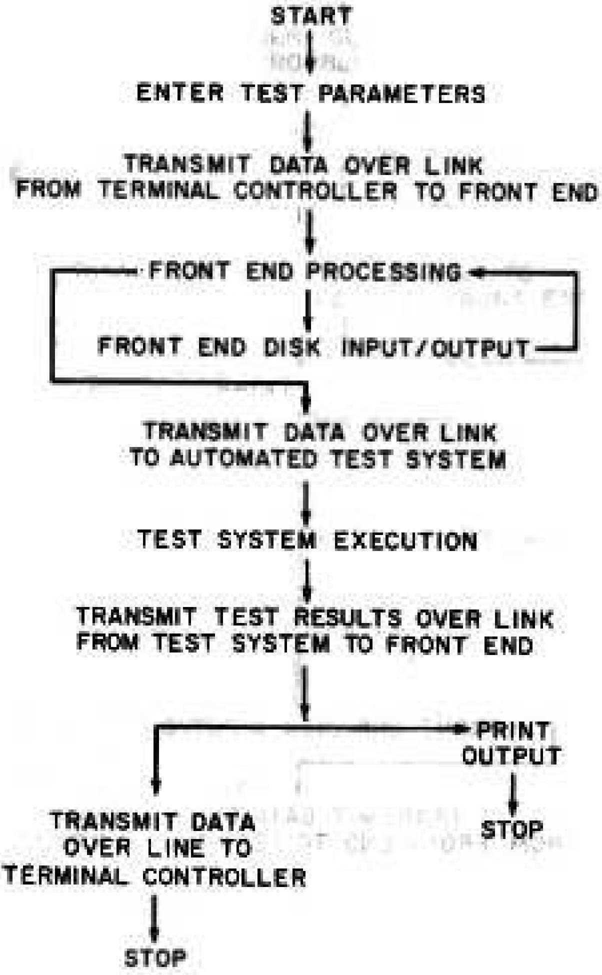

START
ENTER TEST PARAMETERS
TRANSMIT DATA OVER LINK FROM TERMINAL CONTROLLER TO FRONT END
FRONT END PROCESSING
FRONT END DISK INPUT/OUTPUT
TRANSMIT DATA OVER LINK TO AUTOMATED TEST SYSTEM
TEST SYSTEM EXECUTION
TRANSMIT TEST RESULTS OVER LINK FROM TEST SYSTEM TO FRONT END
PRINT OUTPUT
STOP
TRANSMIT DATA OVER LINE TO TERMINAL CONTROLLER
STOP

Figure 8.18. Testing Work Flow