QUEUES / Causes and Cures

by Edward L. Fritz

At one time or another, all of us have stood in a queue. A queue is a waiting line. We have waited in supermarket lines, at the bank, at the Post Office, at a toll booth, or in a restaurant. Frequently we may have thought that somebody should do something to cut down our delays.

Some of us might even be aware of the vast body of literature that exists on the subject of queuing, telephone companies throughout the world have pioneered in the development of many mathematical models. The high quality of telephone service, which we take for granted in our daily lives, is directly attributable to the proper design of trunk lines based on queuing considerations.

Others have built on this solid foundation. However, most of the writing in technical journals is couched in mathematical formulae that are too abstract for immediate application. This is an attempt to clarify the various concepts and to bring to the surface those elements of queuing theories that can be applied to design problems in computers or computer installations.

Some queuing problems are obvious -- lines of people -- such as the examples given above. Other problems are more subtle -- people need a service but wander off when the facility is busy. Some examples of these subtle queues are: a desk calculator used by many people, an office duplicating or reproduction machine, a key punching machine, a computer where programmers wait for debug runs. In each of these cases, if the facility is busy, a person may while away his waiting time at some other activity until the facility is free.

A third class of waiting problems arises when work itself is waiting to be done by people or machines. For example, paper work at a typist's desk, material waiting to be loaded at a warehouse, programs batched at a computer are all queues. Problems awaiting executive decisions or technical solutions also are sitting in queues.
In every one of these examples, delays give rise to some cost -- either direct or indirect. On the other hand, an attempt to minimize delays also involves a cost. Therefore, a realistic solution should balance the delay cost against the outlay needed for changing procedures or adding facilities. I will attempt to provide here, at least an awareness of the problem, a measure of consequences, and an outline of the alternatives for solution.

Ed Fritz is a Principal Analyst in our Washington Scientific Office. He has had more than 16 years experience in systems analysis, reliability studies and operations research.

From 1950 to 1956, he participated in a series of studies for the Civil Aeronautics Authority. These studies concerned traffic landing patterns, enroute air traffic, and terminal airport communication.

From 1956 to 1963 Ed was a consultant at General Electric, specializing in the area of reliability measurement. He was concerned with investigating and measuring the reliability of many hardware systems and subsystems. He developed new techniques for measuring system reliability and assessing the reliability of complex space vechicles.

From 1963 to 1965 Ed headed the Operations Analysis Department at ITT's Information System Division. There he developed criteria for measuring the reliability of the U. S. Navy's Tactical Data System.

Ed's published papers include "Information Theory in Air Traffic Control"; "Empirical Entropy-A Study of Information Flow," Control Systems Laboratory, University of Illinois; "Transient vs. Steady State Delays"; "Bayesian Reliability Estimates," plus a large number of internal working papers and analysis reports at Franklin Institute and General Electric.

Ed holds graduate degrees in Mathematics and Statistics from the University of Michigan. He joined CUC in 1965 as Technical Director in the Washington Office.

In the computing industry, both the monitor system designers and machine users are becoming aware of the need to understand queues. The former, because multi-programming queues arise at several points in the machine (e.g. disc arm, drum access, printer, etc.) and the latter because (as shall be explained later) the goals of efficient use of computers are in conflict with higher thruput of problems. Indeed, as machines get faster and the number of jobs which are to be processed increases (e.g. Time Sharing), computing systems will more and more have to be designed with user queues in mind - like telephone systems.

State of the Art

Queuing models are concerned with systems where "customers" demand service more or less at random. For example, a computer facility may handle an average of, say, fifty programs a day but the number arriving during any interval of time (9:00 -- 9:30) is known only on a probability basis.

In general the most important characteristics of a system with queues are: the average arrival rate (a) and the average service rate (s). Another vital factor is the number of parallel service channels (c). The ratio $\frac{a}{cs} = u$ is defined as the utilization rate per channel.

A couple of examples should clarify the significance of these terms. Suppose that, on the average, a computer facility receives eight programs an hour and, with one computer, can handle an average of ten per hour. Then the utilization rate is $u = \frac{8}{(1)(10)} = .80$. In other words, the computer will be busy 80% of the time. On the other hand, consider another facility with an input rate of 24 per hour but with three identical machines, each of which can handle ten programs per hour. Once again, the utilization rate per computer is $u = \frac{24}{3(10)} = .80$. Each computer is still busy 80% of the time but, as we shall see shortly, the second system provides a better quality of service.

Although the above parameters -- arrival rate, service rate, and number of channels -- define a particular system, the actual queue size depends on several other considerations. For example, any kind of rational scheduling will reduce waiting time over the case of pure random arrivals. Similarly, the more equal the jobs are in length, the shorter the waiting times. The worst kind of service time distribution is the negative exponential in which there are many short service times but a few long ones. This kind of distribution is typical of telephone calls, service at banks, and at computer installations where all jobs are processed on a first-come-first-served basis.

This last case arises when many short assembly and debugging runs are combined with longer production runs. Another factor in assessing waiting time is queue discipline -- whether items are served on a first-come-first-served basis or if priorities are assigned in one of many ways. Finally, the question of single vs. batch processing modifies the behavior of the queues.

Simplified Design Approach

A number of books and articles have been published on these and other variations. However, the greater the detail we go into when specifying a particular system, the more complex the mathematics. And it becomes harder to evaluate all the alternatives. In this article I have selected one mathematical model to illustrate some of the principles of congestion and queuing. At least it should provide a feel for how waiting lines build up. At best it can serve as a rough design tool for modifying or installing a particular system. The basic techniques can be applied to many uses, including computer networks, time sharing, multi-processing, and multi-programming besides better allocation of peripheral equipment.

Description of Model

The model I have selected has the following characteristics:

1. New jobs arrive at random times.
2. Job lengths are exponentially distributed (many short ones and a few long ones).
3. Priorities are given on a first-come-first-served basis.
4. Each job is completed before the next is started by one machine.
5. Any number of machines can be used to provide parallel service of several jobs simultaneously.
6. There is no balking or reneging if too many jobs are in queue.
7. The total number of customers or users is large (20 or more.)

This model was chosen for two sound reasons. First, it results in the worst delays possible for a given configuration, utilization (u) and channels (c). In a sense this permits the worst case design. The second reason is that this model is the most tractable mathematically. Although other models have been, or can be, solved in principle, the evaluation of the formulae is too complex to be included in this general survey of this field. The problems associated with time sharing and partial processing of many programs simultaneously are particularly interesting. But the results, of course, depend on the particular techniques and priorities indigenous to each machine. There is much to be done in this field.

One further simplification is possible with this worst case approach. There is no need to determine the average arrival rate and the average service rate separately and their

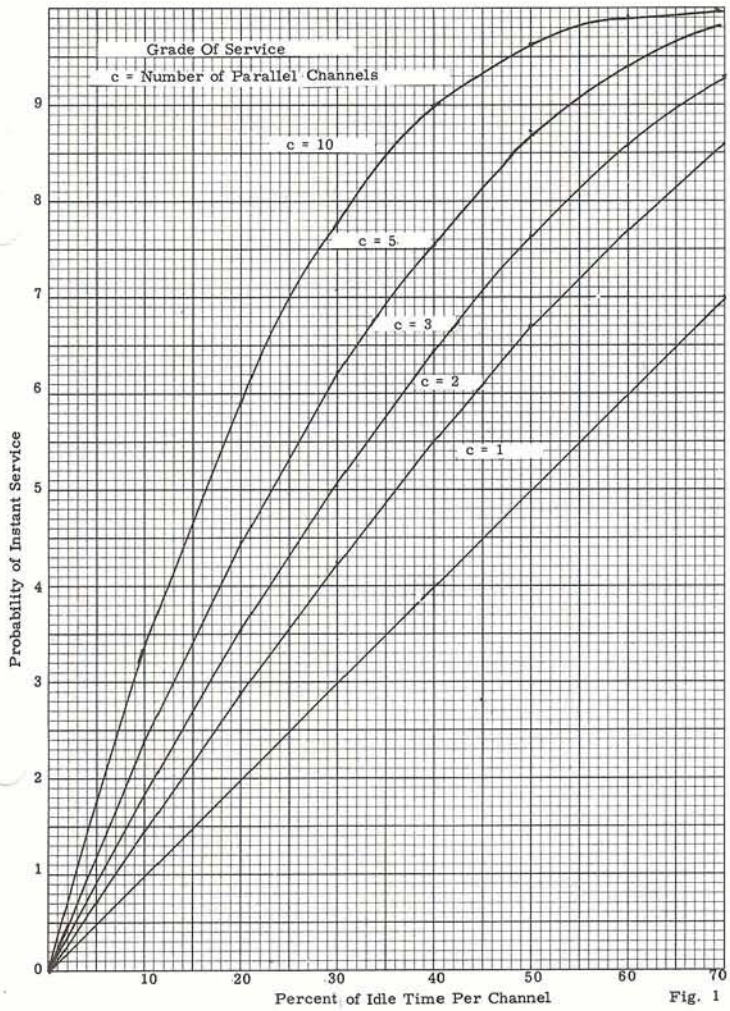

Fig. 1

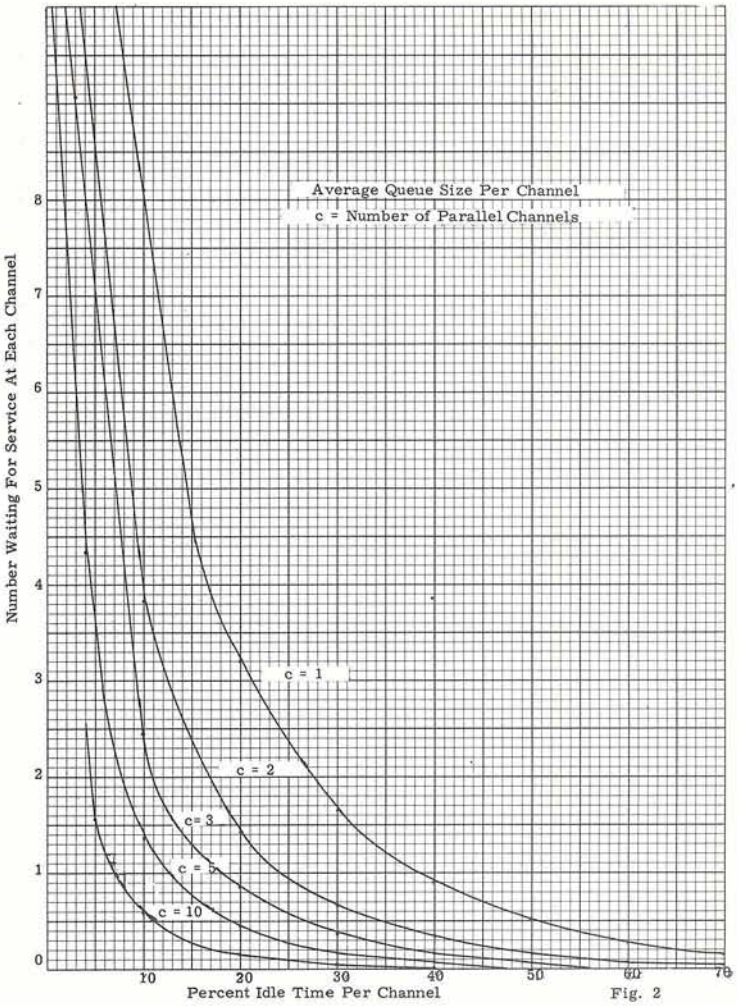

Fig. 2

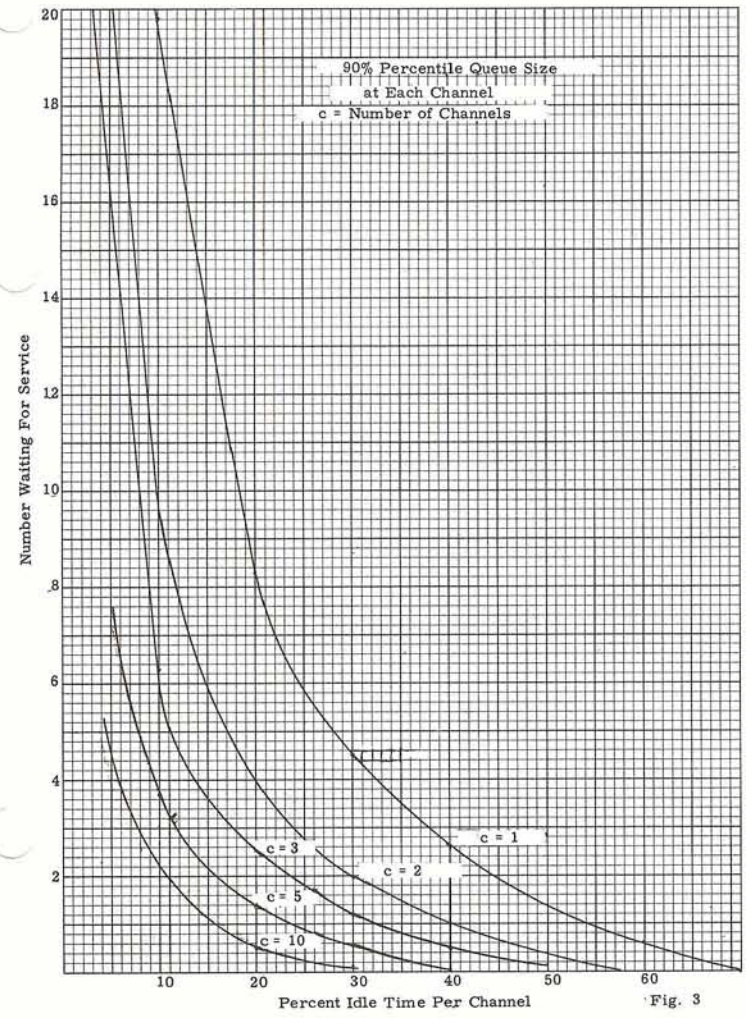

Fig. 3

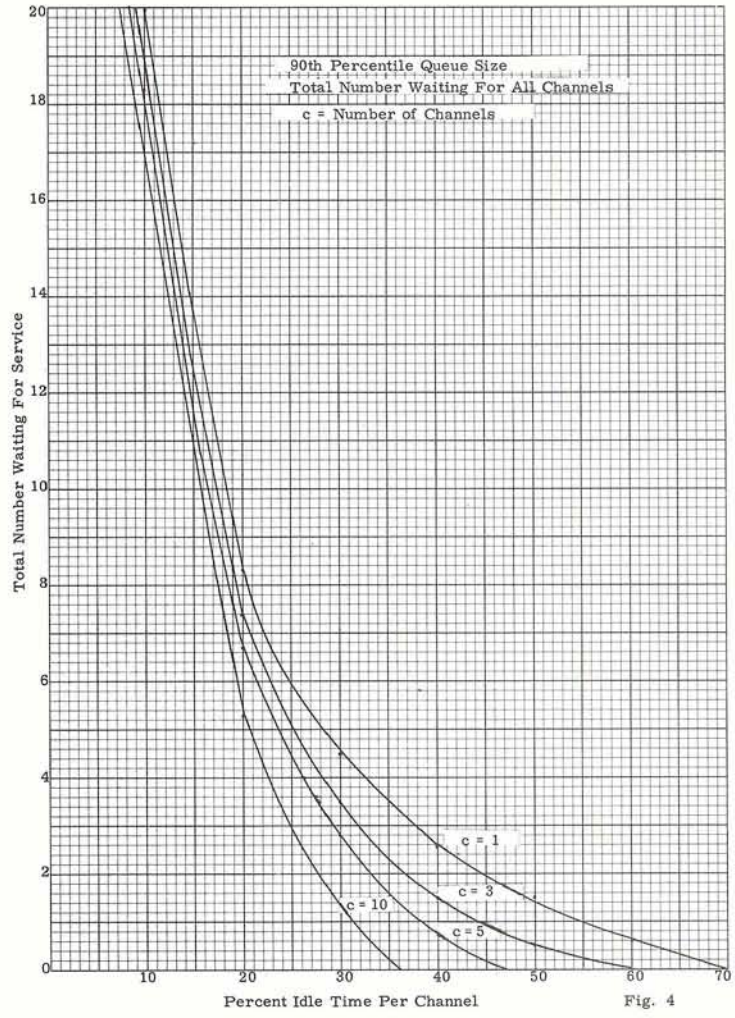

Fig. 4

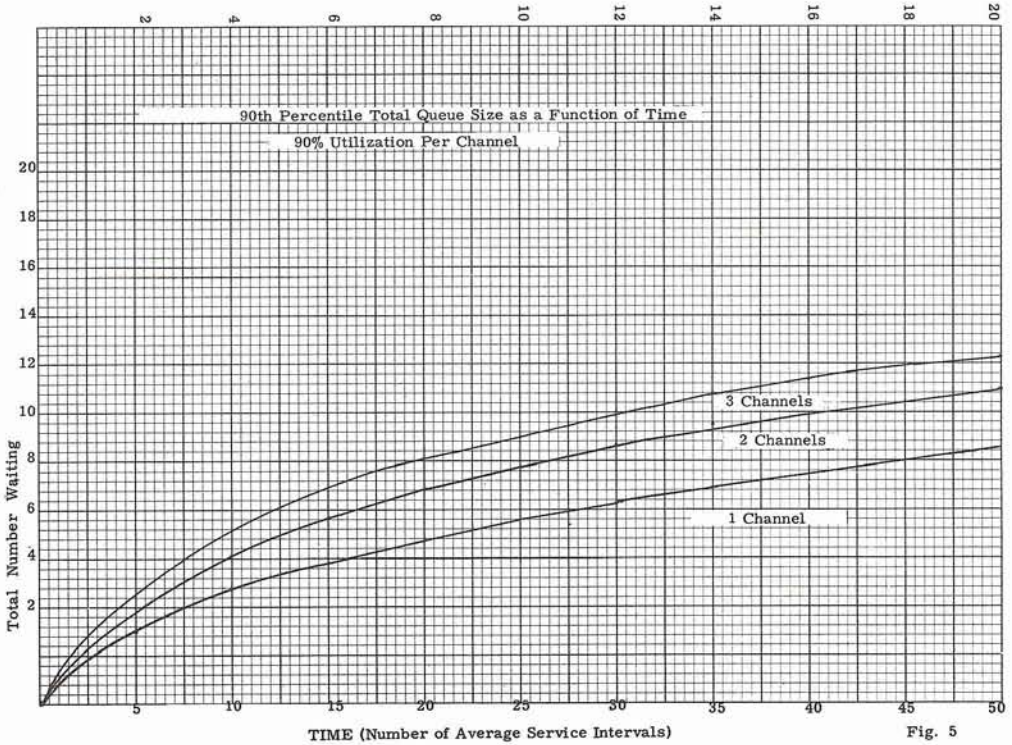

Fig. 5

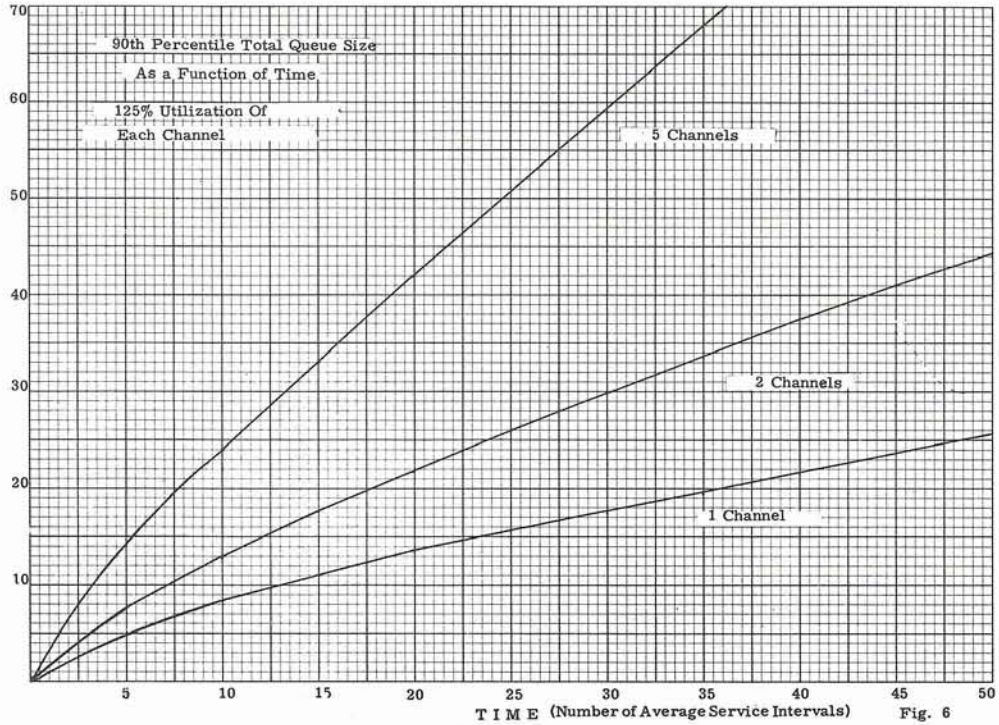

Fig. 6

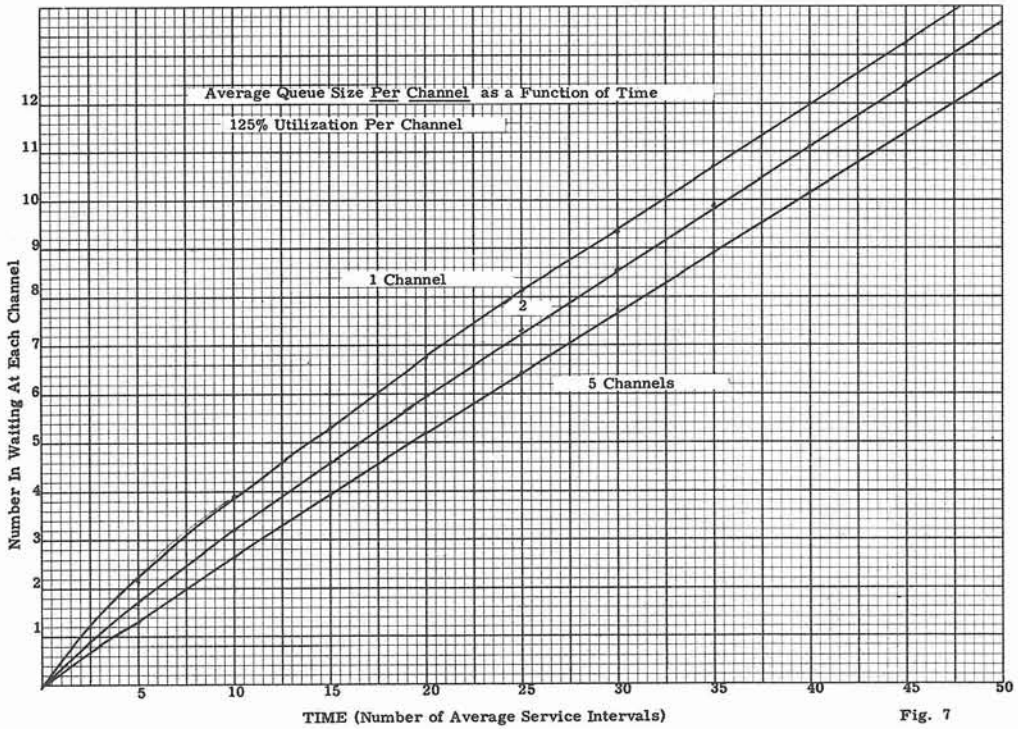

Fig. 7

respective distributions. The channel utilization $u = \frac{a}{cs}$ can be measured directly as the fraction of time each channel is busy. The routine bookkeeping at each installation makes this figure readily available on a shift, daily or weekly basis.

Although the per cent of busy time, or utilization, enters directly into queuing equations, I have chosen to plot the results as a function of idle time to emphasize the fact that any operating system that has elements of randomness in arrivals or service times must provide for some idle time as the cost of giving satisfactory service.

The avowed objective of the manager or owner of any computer is to get 100% utilization. Admittedly, this is a worthwhile goal, which keeps unit costs as low as possible. But, now, what are the consequences of full time or nearly full time usage of any kind of facility with random arrivals? There are a number of measures to describe these effects, and I will now examine them.

Grade of Service

First, let us look at Grade of Service, which is simply the probability that a new job can get instant service. We might ask the question: "How often does an arrival into the system get handled without delay?" For example, how often do you pick up your telephone and get a dial tone right away? The probability of instant service in this case is high. This is because telephone companies design for an excellent grade of service. On the other hand, how often does a programmer walk up to a computer and get instant service? Hardly ever. Computer facilities provide a notoriously poor grade of service because they are almost never idle.

Here, then, is the crux of the problem. In order to provide a high grade of service, a facility has to be idle some of the time! The greater the idle time, the better the grade of service. Also, the cost!

Figure 1 shows grade of service as a function of idle time for our model. This graph illustrates an interesting phenomenon with respect to the number of channels of service. Larger installations are more efficient than smaller ones. For example, suppose a small service bureau with one computer feels that it should provide a 30% grade of service to keep customers happy. That computer, then, will be idle 30% of the time. On the other hand, a larger bureau with enough business for five computers or CPU's can provide the same grade of service with only 13% idle time per computer. We can conclude, therefore, that whenever possible, parallel duplicate services should be provided by a system in preference to separate, but equal, facilities.

Queue Size Per Channel

Another measure of quality of service is the number of jobs waiting at each channel or server. Although the grade of service gives the probability of immediate service, a "customer" may not be too upset, or the cost may not be too great, if only a small delay is incurred.

Figures 2 and 3 show the average waiting line and the 90th percentile queue size, respectively, in front of each channel. Both graphs show the number waiting for service and do not include those being served. Once again, dramatic differences show up between small systems with one or two channels and larger systems. Also, these graphs show rapid deterioration in service (buildup in queues) when idle time falls below 10%.

Total Waiting Line Size

In the proper design of a system we must consider not only the quality of service, but should provide facilities for storing those jobs that are waiting for service. In order to do this we need a measure of how large the total queue can get. Note that Figures 2 and 3 show the number waiting at each channel. An acceptable criterion for a system might be the 90th percentile of total queue size. This measure, as a function of idle time, is shown in Figure 4. A waiting area designed according to this graph would be adequate 90% of the time and 10% of the time we could expect some overflow.

An interesting phenomenon shown in Figure 4 is that the total queue size is not highly dependent on the number of channels. Thus, for an idle time per channel of 10%, we should design the waiting area for between 17 and 20 items. However, slight variations in idle time between 8% and 12% are much more critical to the queue size than the number of channels. This is an apparent refutation of the previous figures which showed a much higher efficiency for larger systems. Actually, though, the efficiency is still there. Consider the previous example of a small service bureau with one computer. If the machine is busy 90% of the time we should provide a waiting area to hold 20 customers. Similarly, the larger bureau with five machines each busy 90% of the time also will need the same waiting area. But the larger bureau is serving five times as many people! Thus, the waiting area per transaction is much less for the larger system.

Transient Queue Buildup

We can readily see from Figures 2 - 4 that as the idle time approaches zero, (or, conversly, as the utilization approaches 100%), the delays tend toward infinity. On the other hand, we have all worked at computer installations where for one or two shifts the computer is indeed busy 100% of the time

with a constant backlog. But in practice the delays are not infinite. At times they seem interminable, but all work does get finished.

How, then, do we justify the mathematical model with reality? The truth is that the model assumes a constant rate of input over an infinite length of time. In practice an installation has a high rate of input over one shift, possibly a lower rate for the second, and third shift if used to work off the backlog from the previous two.

Another factor ameliorating the actual queue size is the practice of "balking" in which a programmer simply stops producing until the machine catches up with him. On the other hand, he may "renege" -- by pulling his work from the queue to do more manual debugging.

I knew of a facility with 28 scientific programmers where the computer was busy 24 hours a day, seven days a week. Each morning the staff worked diligently on the previous night's results until the queue of new work built up at the machine. By the middle of the afternoon it was heartwarming to see the comeraderie and good fellowship which spread among the idle programmers. By Friday the backlog was great enough to allow detailed planning of weekend social activities. The machine room manager, though, took great pride in the fact that by Sunday night the entire backlog was always worked off! He could see no reason for the complaints of poor service -- or for the queue of programmers and engineers at his office demanding priorities! Needless to say, a new computer and a new manager solved the problems.

This article, however, is not concerned with the problems of balking and reneging. Assuming a reasonably well run facility, we should be able to measure the queue buildup over a finite period of time. At Computer Usage Development Corporation's Washington office, we have programmed a set of differential equations describing the transient nature of queues, for arbitrary input or service functions varying with time, for any number of parallel service channels, for the model covered in this article.

For illustrative purposes, we have run two sets of inputs: one where the utilization is fixed at 90% and the other at 125%. For both cases the duration was set at 50 average service times. The principle of measuring time in units of service times keeps the results general enough for many applications. Thus, if the average service time (job length) at one installation is six minutes, the 50 units cover 6x50=300 minutes or five hours. Similarly, if the average service time is 15 minutes, the 50 units cover 15x50=750 minutes or 12.5 hours.

The results are shown in Figures 5, 6, and 7. Figure 5 shows the 90th percentile queue size increasing with time for 90% utilization per channel. We see that by the 50th time unit, the queue has grown to between 12 and 15 jobs. This should be compared with the steady state results of Figure 4 where, for 10% idle time, the expected value is between 18 and 20. Obviously, the queue increases asymptotically to these larger numbers.

Figure 6 shows the same 90th percentile queue size for a utilization of 125%. These lines no longer approach an asymptote. Rather, the queues build up linearly as long as the input rate remains constant.

In Figure 7 we see the average queue per channel. Here also, for 125% utilization, the average increases linearly with time. In particular the slope becomes .25 jobs per channel per unit time. This is to be expected since 125% utilization means that, on the average 1.25 jobs arrive in the time it takes to process one job.

Looking To The Future

As our industry passes from an off-line processing concept -- where programmers leave their decks of cards to be batch-processed at some later time -- to a real time computing concept with programmers conversing directly with computers, the queuing problems will become more acute. Not only will people be more aware of delays as they sit at consoles waiting for responses, but provision will have to be made at the hardware for storing the many requests and assigning priorities.

Much work is still needed in development of models and in the use of these mathematical models for designing hardware and procedures. But whatever form these models take, the principles shown in this article will still apply. To summarize, these principles are: (1) some idle time is necessary to provide a high grade of service; (2) multiple channels of service should be available either with multiple CPU's or the connection of computers by phone lines into a network; (3) whenever possible, the service time should be reduced for on-line operation. This can best be done by handling short jobs on prime time and long production jobs at night or with low priority.

Providing a high quality of service will cost money. However, hardware manufacturers should realize the added benefits and users should balance the added costs against the savings in labor costs. As the "Help Wanted" sections of newspapers testify, the capacity of a computer facility is no longer machine bound. It is limited by the number of programmers and the effective use of those programmers.