CHAPTER 9: **PRIORITY SCHEDULING I**

Up to this point we have concentrated on analyzing the mean throughput rate for multiple resource systems, and the mean throughput rate and mean delay bounds for networks of single resource systems. We saw that for many different types of systems typically *one* single type of resource is limiting the maximum mean throughput rate. The avenues available for improving system performance are to add more resources (perhaps moving the bottleneck resource elsewhere) or *scheduling* the single resource in an effective manner. In this section we will focus on different techniques for scheduling a single resource, in order to meet delay criteria. In practice a system cannot be exercised continuously at its maximum mean rate of completing work, but rather must complete work at some lower rate in order to meet delay criteria of different types. How much below complete utilization this one resource can operate is the subject of this section (and the realm of *queueing theory* as a branch of applied mathematics).

Rather than focusing on simply a mean value analysis, we now worry about other phenomena:

- *fluctuations* about a mean value and
- *correlations* among the fluctuations.

Intuitively, the more regular (the less the fluctuations) or constant and the more predictable the fluctuations (the correlations in the fluctuations), the easier it will be to meet delay criteria, and vice versa. Since more detailed questions are asked, more detailed information must be given to describe system operation. In any queueing system there are three main ingredients:

- a characterization of the arrival statistics of each transaction type
- a characterization of the resources required at each stage of execution of each transaction type
- a policy arbitrating contention for resources (remember there is only a limited amount of each resource type!)

In the previous sections, we used mean values to characterize the arrival statistics (e.g., a rate) and resources required at each stage of execution, for a given policy. In analyzing the mean throughput rate and mean delay in earlier sections, when we fixed the mean time for execution at each stage of each job, we saw that the best performance was obtained when the mean times for each stage of execution were *constant* or *deterministic* while the worst performance was obtained when the fluctuations about the mean times became larger and larger. Here, we expect to see similar phenomena. Remember: all the systems we deal with are *deterministic* in their functional operation, but they are sufficiently complex that we choose a *statistical* or *probabilistic* characterization of the arrival and service statistics, in order to summarize with a very small number of parameters what in fact is a very large number of parameters whose detail may be overwhelming.

9.1 Time Scale of Interest

What time scale is of interest? In order to adequately characterize a system statistically, we expect the measurements we take to stabilize at some (small!) range of values if measurements are carried out over a sufficiently long time interval. How long is long enough? There is no simple answer here. For example, if the disk subsystem is capable of making a disk access every thirty milliseconds, and the processor is capable of doing useful work every ten milliseconds, then if we gather measurements over a time scale one hundred or one thousand times as long as these smallest time intervals, for example, every ten or every thirty seconds, then this is a long time interval during which there is a reasonable possibility that the system has stabilized. On the other hand, it is easy to produce counterexamples in which this need not be the case. The problem is studied in the realm of time series analysis and we will drop it from further consideration here. Our intent is merely to point out that this is a real problem that must be dealt with in actual systems. Queueing systems have both an initial or transient behavior and a long term time averaged behavior. The long term time averaged behavior will occupy all of our

attention here, but the transient behavior is clearly of great interest in many applications. When measurements are presented on any system, always check to see at what point transients have died out and at what point long term time averaged behavior appears to set in. Unfortunately, since we wish to study behavior with congestion, as load builds, transients take longer and longer to die out, and we need more and more data to see the demarcation between the two regimes!

9.2 Workload Arrival Statistics

How do we characterize the arrival statistics? Suppose we observed N arrivals in an interval of duration T time units starting at time zero, and we recorded the arrival time epochs as $t_K, K=1,...,N$ which may possibly be the empty set. One way to characterize the arrival statistics would be by a cumulative distribution function

$$distribution\ \ function = PROB[t_1 \le T_1, \ldots, t_N \le T_N]$$

for each value of N, or the interarrival time distribution for each value of N. In practice, this becomes very unwieldy, so in the next section we introduce additional restrictions on the arrival process that are very often met in practice yet are analytically tractable.

9.2.1 Finite Source Arrival Statistics We first turn to the type of arrival statistics that we used to model clerks at terminals submitting transactions in earlier sections. Since we only have a finite number of clerks and terminals, the population is *finite* and the arrival statistics are due to a *finite set of sources* which leads to the title of this section. The sequence of times from the completion of the last transaction until the submission of the next transaction fluctuate; the time intervals are called *random variables* because they vary and because the fluctuations, while due to many diverse causes, are so complex that we simply summarize all of this by calling them random. Every sequence of time intervals is different from every other, because the precise combination of events leading to that set of times is highly likely to be duplicated exactly.

Example. Ten clerks submit one hundred transactions each to an online transaction processing system. The time interval between the completion of the last transaction and submission of the next transaction is recorded, and the results summarized in the table below:

Table 9.1.One Hundred Transactions/Each of Ten Clerks

Intersubmission Time	*Number of Transactions*
0-2.5 seconds	150
2.5-5.0 seconds	135
5.0-7.5 seconds	110
7.5-10.0 seconds	90
10.0-12.5 seconds	80
12.5-15.0 seconds	65
15.0-17.5 seconds	60
17.5-20.0 seconds	45
20.0-25.0 seconds	75
25.0-30.0 seconds	55
>30 seconds	135

We want to summarize all this information with *one* parameter, the *average* or *mean* intersubmission time. We compute this by multiplying the *fraction* of transactions with a given mean intersubmission time by the *maximum* intersubmission (e.g., 0-2.5 seconds means we assume all jobs had an intersubmission time of 2.5 seconds), and then summing the resultant terms:

$$E(T_{idle}) = 2.5(.150)+5.0(.135)+7.5(.110)+10(.09)+12.5(.08)+15(.065)$$

$$+17.5(.06)+20(.045)+25(.075)+30(.055)+45(.135) = 14.425\ seconds \approx 15\ seconds$$

Note that we have assumed that *all* intersubmissions greater than thirty seconds were *arbitrarily* forty five seconds.

In summary, we can summarize *all* the data by its average or mean value of approximately fifteen seconds, and if we wish to assess sensitivity to this parameter we can change it to ten seconds or to twenty seconds or to whatever value is felt to be appropriate.

We now make even stronger assumptions:

- The sequence of intersubmission times are independent from transaction to transaction and operator to operator (no coffee breaks, no ganging up at the water cooler)
- The sequence of intersubmission times are identically distributed random variables (all operators and transactions lead to identical intersubmission time statistics)
- The intersubmission times are exponentially distributed random variables.

This last statement, the choice of an *exponential* distribution to summarize *all* the intersubmission time statistics, is a key assumption. In words, this says the fraction of intersubmission times that is less than a given threshold, say X seconds, is approximated by an exponential distribution:

$$fraction\ of\ time\ intersubmission\ time\ interval \leq X =$$

$$1 - exp\,[-X/E\,(T_{idle})] = 1 - e^{-\lambda_{idle} X} \qquad \lambda_{idle} \equiv \frac{1}{E\,(T_{idle})}$$

To test this goodness of fit, the figure below shows a quantile quantile plot of empirical or data quantiles versus exponential model quantiles.

Figure 9.1.Empirical Quantiles vs Exponential Model Quantiles

Since the plot is approximately a straight line, the goodness of fit is felt to be acceptable, and we can use the exponential model with as much confidence as we place in our data and its analysis.

The fraction of time one clerk is idle less than or equal to X is given by

$$PROB\,[T_{idle}\ for\ one\ clerk \leq X\,] = 1 - exp\,[-X/E\,(T_{idle})] = 1 - e^{-\lambda_{idle} X}$$

The probability or fraction of time that we have J submissions by N clerks in an interval of duration T is given by

$$PROB\,[J\ submissions\ by\ N\ clerks\ in\ interval \leq X\,]$$

$$= \begin{bmatrix} N \\ J \end{bmatrix} \left[1 - exp\,(-X\,\lambda_{idle})\right]^{J} \left[\exp(-X\,\lambda_{idle})\right]^{N-J} \qquad J=0,1,...,N$$

Many times the moment generating function of a distribution is easier to work with (analytically or numerically) than the actual distribution. Here the moment generating function is given by

$$E[X^J] = \sum_{J=0}^{N} Y^J \, PROB[J \ submissions \ by \ N \ clerks] = \left[exp(-X\lambda_{idle}) + Y(1 - exp(-X\lambda_{idle})) \right]^N$$

This can be differentiated to find the mean number of submissions in a given time interval, and hence the mean arrival rate:

$$\frac{d}{dY} E[Y^J] \Big|_{Y=1} = E(J) = N[1 - exp(-\lambda_{idle} X)]$$

The total mean arrival rate of work is thus given by the mean rate at which each clerk submits work multiplied by the mean number of clerks in the idle state, reading, thinking, typing, and so on:

$$total \ mean \ arrival \ rate = mean \ number \ of \ idle \ clerks \times \lambda_{idle}$$

As we add more and more clerks, the mean number *idle* (as well as the mean number waiting for response) will grow without bound over any threshold we set. What we want to do is fix the *total mean arrival rate* and increase the total number of clerks; this means the total mean number of idle clerks will grow without bound, while the total mean arrival rate is fixed, so the *mean idle time per transaction* must also grow without bound:

$$N \times \lambda_{idle} = \frac{N}{E(T_{idle})} = total \ mean \ arrival \ rate \equiv constant = \lambda$$

If we do this, we see that the mean number of arrivals in an interval of duration X seconds is given by

$$\lim_{N \to \infty} \begin{bmatrix} N \\ J \end{bmatrix} \left[1 - exp(-X\lambda_{idle}) \right]^J \left[\exp(-X\lambda_{idle}) \right]^{N-J} = \frac{(\lambda X)^J}{J!} e^{-\lambda X} \quad J = 0,1,2,...$$

Note that each terminal or source is contributing less and less arriving work, because the individual mean arrival time is growing without bound, and hence we call this the *infinite* population limit ($N \to \infty$) of the finite population arrival process, the so called *Poisson* arrival process, which is the subject of our next section.

9.2.2 Infinite Source Poisson Process Now we assume the interarrival times are independent identically distributed exponential random variables, or, that the arrival statistics are *Poisson* distributed. This means that the interarrival times obey the following relationship:

$$PROB[t_{K+1} - t_K \le X] = 1 - exp(-X/E[T_A]) = 1 - e^{-\lambda X} \quad K = 1,2,3,...,N-1$$

$$E[t_{K+1} - t_K] = E[T_A] = \frac{1}{\lambda} \quad K = 1,2,...,N-1$$

Numerous real life situations can in fact be adequately modeled by Poisson statistics. Why should this be a reasonable fit to actual data? Because whenever there is a superposition of a number of independent sources of arrivals, no one of which dominates, then under a variety of assumptions it can be shown that as the number of sources approaches infinity the superposition or sum of all the arrival streams converges to a Poisson process (this is the so called generalized *Central Limit Theorem* of probability theory). Another reason why the Poisson process matches actual data is that it has two properties that are often met in practice:

- the number of arrivals in disjoint time intervals are *independent* random variables
- the number of arrivals is proportional to the duration of the observation time interval

These are properties that are *unique* to the Poisson process, and for all these reasons make it a worthy candidate first cut model of arrival statistics in many applications.

What are some analytic properties of the Poisson process that might be useful in modeling?

- the superposition or sum of Poisson processes is Poisson, with mean arrival rate being the sum of the individual arrival rates: if we add two processes, the rates add and the new process is Poisson
- the randomized thinning of a Poisson process is a Poisson process, whereby at each arrival epoch we flip a coin and with probability P include the arrival epoch while with probability $(1-P)$ we discard it, i.e., we thin the arrival stream, with mean arrival rate P times that of the original process's arrival rate

The moment generating function of the interarrival time distribution is given by

$$E[exp(-z(t_{K+1}-t_K)) = \int_0^\infty e^{-zX} d_X[1-e^{-\lambda X}] = \frac{\lambda}{\lambda + z}$$

and this can be used to find moments of all integral order by the identity

$$E[{T_A}^K] = (-1)^K \frac{d^K}{dz^K}\left[\frac{\lambda}{\lambda + z}\right]\Big|_{z=0} \qquad K=1,2,3,...$$

For example, the second moment of the interarrival time distribution of a Poisson process is given by

$$E({T_A}^2) = \frac{2}{\lambda^2} \rightarrow var(interarrival\ \ time) = E({T_A}^2) - E^2(T_A) = E^2(T_A) = \frac{1}{\lambda^2}$$

A related quantity is the number of arrivals in a time interval of duration T, denoted $N(T)$. Assuming the arrivals are Poisson we see

$$PROB[N(T) = K] = \frac{(\lambda T)^K}{K!}\ exp(-\lambda T) \qquad K=0,1,2,3,...$$

The moment generating function is given by

$$E[X^{N(T)}] = \sum_{K=0}^{\infty} X^K e^{-\lambda T}\frac{(\lambda T)^K}{K!} = e^{-\lambda T(1-X)}$$

This can be differentiated to find all factorial moments using this identity:

$$E[N(T)[N(T)-1][N(T)-2]...[N(T)-K+1]] = \frac{d^K}{dX^K} e^{-\lambda T(1-X)}\Big|_{X=1}$$

This has mean value

$$E[N(T)] = \lambda T$$

while the second factorial moment is given by

$$E[N(N-1)] = (\lambda T)^2$$

9.2.3 General Interarrival Time Statistics What if the interarrival time distribution is arbitrary, i.e., not finite or infinite source processes? One way to characterize the interarrival time distribution is by its first two moments, for example. One measure of the fluctuations in the interarrival time sequence is to measure the variance in units of the mean interarrival time (squared), with the result called the squared coefficient of variation:

$$squared\ coefficient\ of\ variation = \frac{var(interarrival\ \ time)}{E^2(interarrival\ \ time)}$$

Three cases arise:

$$squared\ coefficient\ of\ variation = \begin{cases} <1 & more\ regular\ than\ Poisson \\ 1 & Poisson \\ 1 & more\ irregular\ than\ Poisson \end{cases}$$

9.2.4 Message Switching System A message switching system must handle two different types of messages. Twenty per cent of the messages have a mean interarrival time of 320 microseconds, while eighty per cent of the messages have a mean interarrival time of 1024 microseconds. What is the squared coefficient of variation? First we calculate the mean interarrival time:

$$mean\ interarrival\ time = E(T_A) = 0.20\times 32 + 0.80\times 1024 = 825.6\ microseconds$$

Next, we calculate the variance of the message interarrival time distribution:

$$variance\ of\ message\ interarrival\ time\ distribution = var(T_A)$$
$$= 0.20\times[32 - E(T_A)]^2 + 0.80\times[1024 - E(T_A)]^2$$
$$= 157450.24\ microseconds^2$$

Finally, we calculate the squared coefficient of variation for the message interarrival time distribution:

$$squared\ coefficient\ of\ variation = \frac{var(T_A)}{E^2(T_A)} = \frac{157450.24}{(825.6)^2} = 0.23099$$

This shows that the fluctuations are not as severe as would be encountered with an exponential message interarrival time distribution. In practice, we might choose to be *pessimistic* (fluctuations can presumably only make things worse) by using an exponential interarrival time distribution rather than using a constant interarrival time distribution.

9.2.5 Additional Reading

[1] M.B.Wilk, R.Gnanadesikan, *Probability Plotting Methods for the Analysis of Data*, Biometrika, **55** (1), 1-17 (1968).

[2] H.Heffes, *A Class of Data Traffic Processes--Covariance Function Characterization and Related Queuing Results*, Bell System Technical Journal, **59** (6), 897-929 (1980).

9.3 Service Time Distribution

The sequence of service or processing times can be characterized by their joint distribution. Rather than do so, we assume the processing times are independent identically distributed random variables, because in many cases this is a reasonable match of data, and because it is analytically tractable and allows many sensitivity studies to be readily performed.

9.3.1 Exponential Service Time Distribution A program executes one hundred times on the same hardware configuration with different inputs. The following statistics summarize how long the program executed:

Table 9.2.Execution Time Summary

0-1000 machine cycles	28 runs
1001-2000 machine cycles	21 runs
2001-3000 machine cycles	16 runs
3001-4000 machine cycles	10 runs
4001-5000 machine cycles	9 runs
5001-10,000 machine cycles	13 runs
10,001-15,000 machine cycles	2 runs
>15,000 cycles	1 run
Total	100 runs

We wish to summarize all this data with *one* easy to work with statistic, such as the mean number of machine cycles per run. Here the mean number of machine cycles executed per run is roughly 6,630 machine cycles (check this!). We might use 5,000 to 7,500 machine cycles per job to bracket this estimate in further analysis. What about the distribution of machine cycles per job?

If we test this data against an exponential distribution model using a quantile-quantile plot, the results are plotted in the following figures.

Figure 9.2.Empirical Quantiles vs Exponential Model Quantiles

Since the graph is approximately a straight line, the goodness of fit of the data to the exponential distribution model is felt to be adequate.

Because of this data analysis, we assume that the distribution of machine cycles can be summarized by

$$PROB[execution\ \ time \le X] = 1 - exp(-\mu X) = 1 - exp(-X/E(T_S))$$

$$E(T_S) \approx \frac{6{,}630\ machine\ \ cycles}{machine\ \ clock\ \ rate}$$

Hence, the mean rate of executing work is μ jobs per unit time, while the mean time per job to completely execute it is $1/\mu = E(T_S)$. The moment generating function of this distribution is given by

$$E[e^{-zT_S}] = \frac{\mu}{\mu + z} = \frac{1}{zE(T_S) + 1}$$

The second moment and variance are given by

$$E[T_S{}^2] = 2E^2[T_S] \quad var(T_S) = E^2(T_S)$$

9.3.2 Constant Service Time Distribution If all the runs for a given program require virtually the same amount of time, irrespective of the input, the service times are deterministic or constant, and

$$PROB[execution\ \ time\ \ \le X] = \begin{cases} 1 & X > E(T_S) \\ 0 & X \le E(T_S) \end{cases}$$

The moment generating function for this distribution is

$$E[e^{-zT_S}] = e^{-zE(T_S)}$$

The second moment and variance is given by

$$E[T_S{}^2] = E^2[T_S] \quad var(T_S) = 0$$

9.3.3 Erlang Service Time Distribution Suppose that we have a pipeline of K processors, with each processor executing only one program. Each program and each processor is identical. The operation of a single program on a single processor is measured, and is found to be exponentially distributed. Thus, the total execution time is the sum of the execution time of each stage:

$$T_{S,total} = \sum_{J=1}^{K} T_{S,J}$$

Since each stage is identical, the total mean execution time is K times the mean execution time for any one stage:

$$E(T_{S,J}) = \frac{E(T_{S,total})}{K} \qquad J=1,...,K$$

The moment generating function for the total time to execute a job is:

$$E[e^{-zT_{S,total}}] = \left[\frac{1}{\frac{zE(T_{S,total})}{K}+1}\right]^K \qquad K=1,2,3,...$$

This is called an *Erlang* distribution in honor of the great Danish teletraffic engineer who laid the foundations for much of modern queueing theory and analysis of congestion.

EXERCISE. Check that this has mean $E(T_S)$ for all K.

For $K=1$ this is the exponential distribution, while for $K>1$ this is a more complicated expression. The second moment of this distribution is

$$E[T_S{}^2] = E^2(T_S)\frac{K+1}{K}$$

and hence the squared coefficient of variation is given by

$$squared\ coefficient\ of\ variation = \frac{1}{K}$$

For $K=1$ this is the exponential distribution, while for $K\to\infty$ this is approaching the constant or deterministic distribution, and hence models fluctuations inbetween these extremes.

9.3.4 Hyperexponential Service Time Distribution A processor executes any one of N types of jobs. Once a job begins execution, it runs to completion. Measurements are gathered on the system in operation.

- The fraction of jobs of each type are measured; $F_K, K=1,...,N$ denotes the fraction of jobs that were executed that were type K
- The execution time statistics of each job are measured, and are felt to be adequately modeled for each and every job type by an exponential distribution but with a different mean depending upon the job type: $E(T_{S,K}), K=1,...,N$ denotes the mean execution time of job type K

This type of distribution is called the *hyperexponential* distribution which is a mixture or sum of two or more exponential distributions.

The chart below shows one method for generating hyperexponential random variables: For two distributions, we see

$$PROB[T_S \le X] = F_1(1-e^{-X/E(T_{S,1})}) + F_2(1-e^{-X/E(T_{S,2})}) \qquad F_1 + F_2 \equiv 1$$

If we fix the mean, we can make the squared coefficient of variation greater than one, and hence model fluctuations greater than those encountered for the exponential distribution case.

Exercise: Find the moment generating function of the hyperexponential distribution.

Solution: >From the definition, we see

Figure 9.3.Hyperexponential Random Variable Generation

$$E[exp(-zX)] = \int_0^\infty exp(-zX)dG(X)$$

$$= \sum_{K=1}^{N} F_K \int_0^\infty exp(-zX)\frac{1}{E(T_{S,K})}exp(-X/E(T_{S,K}))dX = \sum_{K=1}^{N} F_K \frac{1}{zE(T_{S,K})+1}$$

9.3.5 Hypoexponential Distribution A job is decomposed into N tasks, with each task executed by a single processor. Measurements are gathered on

- The fraction of arriving jobs due to each type of task, denoted by $F_K, K=1,...,N$
- The execution time statistics of each task which are exponentially distributed with mean $E(T_{S,K}), K=1,..,N$ for stage K

The chart below summarizes work flow.

Figure 9.4.Hypoexponential Distribution Flow Chart

This type of execution time statistics is called the *hypoexponential* distribution which generalizes the Erlang distribution. One way to generate random variables from this type of distribution is to first generate an exponential random variable, and then allow a fraction F_1 of jobs to have this distribution while the rest of the jobs require not only this service but additional service which is generated from a second exponential distribution with possibly different mean, and we can repeat this process again and again.

Exercise: Show the Erlang distribution is hypoexponential.

Solution: A random variable from the Erlang distribution can be generated by using N exponential random number generators each with the same mean in a pipeline, with the output from one stage feeding the input to the next stage.

9.3.6 Single Link Flow Control One transmitter receives messages from a variety of sources and transmits them over a data link to a single receiver. The following steps are involved in message transmission:

[1] The transmitter adds source and destination header to the message, adds start and end of message delimiters, adds an appropriate cyclic redundancy check to the message, and transmits the message; this requires a time interval $T_{transmitter}$

[2] The receiver buffers the message, strips off the start and end delimiters, checks the cyclic redundancy check to see if errors might be present, passes the message on to the proper destination, and if all is correct transmits an acknowledgement of proper transmission to the transmitter; this requires a time interval $T_{receiver}$

[3] The transmitter processes the acknowledgement, and flushes the message from its buffer

The transmitter and receiver can operate at widely disparate speeds:

- The transmitter might be an electromechanical terminal and the receiver a computer
- The transmitter might be an intelligent disk controller and the receiver a microprocessor controlled electronic funds transfer automatic teller

In order to insure that no messages are lost due to speed mismatching, because the *receiver* has limited and *finite* buffering or storage of messages, a maximum of **B** unacknowledged messages can be transmitted from the transmitter to the receiver; beyond this point, the transmitter waits to receive an acknowledgement. This is called *flow control* because the flow of data over the link is paced or controlled by this mechanism.

We now consider the case where the propagation effects are negligible compared to the transmitter and receiver message execution times. This will be the case in a local area network, with terminals and computers interconnected over a distance of a few kilometers or less, for example. Since we only have *one* transmitter and *one* receiver, the greatest amount of concurrency or parallelism possible is to have both resources executing messages. Thus, we consider the case where at most two unacknowledged messages can be outstanding (B=2) at the transmitter at any one time. For this special case, we also show an illustrative timing diagram of system operation.

Note that after the initial idle to busy transient condition, the transmitter and receiver are very strongly coupled in their operation. A race condition can occur if the transmitter finishes before the receiver or the receiver before the transmitter and two messages have to be acknowledged. If we ignore the startup transient, the time required to transmit a message, denoted by $T_{message}$, by inspection of the figure, is given by

$$T_{message} = max(T_{trans}, T_{rec}) + T_{rec}$$

The mean message transmission time is given by

$$E(T_{message}) = E[max(T_{trans}, T_{rec})] + E(T_{rec})$$

Figure 9.5.Illustrative Operation with Negligible Propagation Time

The message handling time is not simply the sum of the individual message handling times. What is the distribution of the message handling time distribution? To find this, we need to find the distribution of the maximum of the transmitter and receiver message handling time distributions:

$$PROB[max(T_{trans}, T_{rec}) \leq X] = PROB[T_{trans} \leq X] PROB[T_{rec} \leq X]$$

On the other hand, rather than work with this complicated expression directly, we might be satisfied with simply *bounds* on the message handling time:

$$max[E(T_{trans}, T_{rec})] \leq E[max(T_{trans}, T_{rec})] \leq E(T_{trans}) + E(T_{rec})$$

Check this! How do we interpret these two bounds?

- The lower bound says that the slower of the transmitter and receiver will be the system bottleneck
- The upper bound says that if fluctuations are sufficiently great about the mean values, the mean message handling time will be approximately the same mean time as if the receiver could only buffer *one* message at a time, with no parallelism or concurrency possible

Here is a different way of understanding this phenomenon:

- The transmitter can be much slower than the receiver:

  $$E(T_{trans}) \gg E(T_{rec})$$

  and hence there will never be any queueing at the receiver, or the receiver can be much slower than the transmitter:

  $$E(T_{rec}) \gg E(T_{trans})$$

  and hence there will always be two messages at the receiver. This case is called *speed mismatch*
- Fluctuations about the mean transmitter and receiver times can be severe:
  - If the transmitter and receiver message service times are *constant*, then

    $$E(T_{message}) = max[E(T_{trans}), E(T_{rec})] + E(T_{rec})$$

  - If the transmitter and receiver message service times are *exponential* random variables, then

    $$E(T_{message}) = \frac{E(T_{trans}) E(T_{rec})}{E(T_{trans}) + E(T_{rec})} + E(T_{rec})$$

The figures below plot the ratio of the mean of the maximum of the transmitter and receiver message handling times divided by the single message at a time, assuming the transmitter and receiver have *identical* distributions with identical squared coefficients of variation.

Figure 9.6.Mean Throughput Gain of Double vs Single Buffering

When the speed of the transmitter and receiver are roughly equal, and the squared coefficient of variation is close to zero, then the gain approaches fifty per cent. When the speeds of the transmitter and receiver are mismatched by a factor of two or more, or the fluctuations become significant (squared coefficient of variation greater than one), or both, the gain is roughly ten per cent or less.

9.4 Resource Allocation Policy

At each arrival epoch and each completion epoch, a decision must be made for which task(s) are processed. What are some of the means for resolving contention for a processor?

- At each arrival epoch, we could use the arrival time as a priority index. The smaller the index, the higher the priority leads to the policy of service in order of arrival, or first come, first served.
- At each arrival epoch, we could use the arrival time as a priority index, but now the larger the index the higher the priority! If we decide that we will execute jobs to completion once they begin to execute, i.e., to execute *nonpreemptively*, then our work discipline is service in reverse order of arrival, last come, first serve, nonpreemptive. If we decide that we will make scheduling decisions at job arrival instants, i.e., to execute *preemptively*, then we could choose to preempt a job and either *resume* processing at the point of interruption or *repeat* processing from the beginning, or any of a number of other points.
- At each arrival epoch, we could use a given label on the job as a priority index. Jobs would be executed according to priority index; jobs with the same index would be executed in order of arrival. At least two options are available, *preemptive resume* scheduling where higher priority jobs interrupt lower priority jobs upon arrival and execution is resumed at the point of interruption, and *nonpreemptive* scheduling where once a job begins to execute it runs to completion.
- At each arrival epoch, we could use both the arrival time and a second quantity which is the desired execution time window of that job to determine priority: we simply add the arrival time to the window, and execute jobs according to highest priority. This is called *deadline* scheduling because the arrival time plus window is called the *deadline* for that job.
- At each arrival epoch, we could use the processing time of the job to determine its priority. If we schedule jobs nonpreemptive, then one such rule is to execute that job with the *shortest processing time* at all times; if we schedule jobs preemptive resume, then one such rule is to execute that job with the *shortest remaining processing time* at all times.

Many more policies are known, as well as variations on those above. We will not deal with all of these, but only wish to give the reader some idea of how rich the policy space in fact is. One way of classifying these policies is by the labels *static* and *dynamic:* a *static* policy depends only upon the attributes of a job that do not change with time, while a *dynamic* policy does allow the priority or urgency of a job to depend on time.

9.5 Measures of Congestion

Broadly speaking, there are two types of congestion measures, those oriented toward the *customer* and those oriented toward the *system.* For each type, we might associate a cost, and then attempt to trade off among them: as we improve customer oriented measures, system oriented measures degrade, and vice versa!

Customer oriented criteria deal with the *mean throughput rate* and the *delay statistics* for each type of task. We characterize delay by

- queueing time or flow time or time in system of a task, denoted T_Q
- waiting time, the time interval from arrival until a task first receives service, denoted T_W

System oriented criteria deal with

- *mean number of jobs in execution in the system,* defined as

$$mean\ number\ of\ executing\ jobs = \frac{mean\ service\ time}{mean\ interarrival\ time}$$

  which follows from Little's Law

- utilization, defined as

$$utilization = fraction\ of\ time\ resource\ busy$$

- distribution of number of tasks in the system (at arrival epochs, completion epochs, or *arbitrary* time epochs)

This list is not complete. Our goal will be to calculate these different measures given certain arrival statistics, service time statistics, and policies for arbitrating contention.

9.5.1 Additional Reading

[1] R.B.Cooper, **Introduction to Queueing Theory,** Chapters 1-3, Macmillan, NY, 1972.

[2] H.Kobayashi, **Modeling and Analysis: An Introduction to System Performance Evaluation Methodology,** Chapter 2, 3.1-3.5, Addison-Wesley, Reading, Mass. 1978.

[3] L.Kleinrock, **Queueing Systems, Volume I: Theory,** Chapters 1-2, Wiley, NY, 1975.

9.6 Approximation of the Inverse of a Laplace Transform

We have already introduced and used the concepts of moment generating function and Laplace transform. These transforms are often easier to manipulate and work with, rather than working directly with probability distributions. Furthermore, since we are demanding additional distributional information concerning the workload, we would like to get more out of this input than simply a mean value: what fraction of the time does a transaction take longer than T seconds to spend either waiting or executing? what point in time in system will result in ninety per cent or ninety five per cent or ninety nine per cent of the transactions being completed? This suggests numerical methods for approximating the inverse of the moment generating function or Laplace transform.

The properties that we wish to preserve with our numerical methods are

- Nonnegativity--Certain types of approximations result in *negative* probabilities in exactly the region of interest, the *tail* of the distribution, the point at which ninety five per cent of the jobs have been completely executed; our goal here is to have a nonnegative class of approximations

- Monotonicity--Certain types of approximations, such as those based on Fast Fourier Transform methods, result in oscillations and waves in the region of interest (not surprising, if the approximation is a sum of sinusoidal waves); we wish to preserve monotonic behavior

In summary, we will approximate a distribution function by another distribution function.

9.6.1 Description of Basic Approximating Algorithm Consider a function $g(x)$ for which the Laplace transform, denoted $\tilde{g}(z)$, defined by

$$\tilde{g}(z) = \int_0^\infty e^{-zx} g(x) dx$$

exists for all $Re(z)>\beta$ where β is the *abscissa of convergence* of the transform. We will assume that $g(x)$ drops off exponentially fast in x, which will occur with the problems we will encounter.

$$\lim_{x\to\infty} g(x)e^{\gamma x} = \begin{cases} \infty & \gamma<\beta \\ constant & \gamma=\beta \\ 0 & \gamma>\beta \end{cases}$$

We now define a sequence of linear functionals that will approximate $g(x)$, denoted by $L_n, n=0,...,\infty$, given by

$$L_n[g(x)] = g_n(x) = \frac{(-1)^n}{n!} z^{n+1} \frac{d^n \tilde{g}(z)}{dz^n} \Big|_{z+\frac{n+1}{z}}$$

It can be shown that

$$\lim_{n\to\infty} g_n(t) = g(t)$$

The zeroth order approximation is given by

$$L_0[g(x)] = z\tilde{g}(z) \Big|_{z=\frac{1}{x}}$$

while the first order approximation is given by

$$L_1[g(x)] = -z^2 \frac{d\tilde{g}(z)}{dz} \Big|_{z=\frac{2}{x}}$$

The reason for choosing this type of approximation is that the resulting approximation is nonnegative. Other methods, such as those based on Fast Fourier Transform techniques, do not preserve positivity. This is not free: if we go from the nth order approximation to the 2nth order approximation, the error will only halve, while using other methods (such as those based on Fast Fourier Transform techniques) result in the error being quartered.

9.6.2 An Example To illustrate all of this, we try a probability distribution function of the form:

$$PROB[T>x] = g(x) = \tfrac{1}{2}e^{-x} + \tfrac{1}{2}e^{-3x}$$

This is convenient and easy to work with analytically, to illustrate our points. The Laplace transform of this function is given by

$$\tilde{g}(z) = \int_0^\infty g(x)e^{-zx} dx = \tfrac{1}{2}\left[\frac{1}{z+1} + \frac{1}{z+3}\right]$$

The abscissa of convergence here is one, i.e., the minimum of three and one.

The zeroth and first order approximation to $g(x)$ is given by

$$g_0(x) = \tfrac{1}{2}\left[\frac{1}{1+x} + \frac{1}{3x+1}\right] \quad g_1(x) = \tfrac{1}{2}\left[\frac{1}{(\frac{1}{2}x+1)^2} + \frac{1}{(3x/2+1)^2}\right]$$

The tables below summarizes numerical studies as a function of approximation parameters:

Table 9.3. α=0

x	g	g_{50}	g_{100}
2	0.06891	0.07203	0.07048
4	0.00916	0.01064	0.00990
8	0.00017	0.00030	0.00023

Table 9.4. α=1

x	g	$g_{50,1}$	$g_{100,1}$
2	0.06891	0.06911	0.06901
4	0.00916	0.00916	0.00916
8	0.00017	0.00017	0.000017

Since the abscissa of convergence of the Laplace transform of $g(x)$ is one, α=1 is the appropriate value to use here.

The error is halved in going from fifty to one hundred terms, as expected. The effect of α=1 is greater for larger values of t since the exponential term which is being tracked becomes dominant.

A different type of example is shown in the figure below. The total mean arrival is fixed at 0.5 arrivals per second, with each arrival requiring a mean service of one second. Jobs are executed in order of arrival. The final parameter is the second moment of the service time distribution; this allows fluctuation about the mean.

Figure 9.7.Fraction of Time $T_Q > X$ vs X

For example, ninety per cent of the jobs are executed within four seconds of their arrival if there is *no* fluctuation in the service time distribution, while for the exponential distribution ninety per cent of the jobs are executed within six seconds of their arrival.

9.6.3 Additional Reading

[1] D.Jagerman, *An Inversion Technique for the Laplace Transform with Application to Approximation,* Bell System Technical Journal, **57** (3), 669-710(1978).

9.7 Kendall's Queueing System Notation

Kendall has introduced a notation for characterizing queueing systems:

ARRIVAL/SERVICE/NUMBER OF SERVERS/CAPACITY

These terms are as follows:

- *ARRIVAL* --the interarrival time distribution
- *SERVICE* --the service time distribution
- *NUMBER OF SERVERS* --the number of processors or servers
- *CAPACITY* --the total system capacity for tasks (if this is infinite, this term is often omitted)

The following abbreviations are often used to characterize these different interarrival and service time distributions:

- *M* --exponential (Markovian) distribution
- *D* --deterministic or constant distribution
- E_K --Erlang-K distribution, a type of gamma distribution
- *H* --hyperexponential distribution (linear combination of two or more exponentials)
- *G* --general or arbitrary distribution (as distinct from the above highly structured distributions)

Many other abbreviations have crept into use than just those above. Here are some examples of this nomenclature:

- *M/M/*1--a single server queue with exponential interarrival times and exponential service times
- *M/G/*1--a single server queue with exponential interarrival times and arbitrary or general service times
- $G/E_K/3/7$--a three server queue with capacity seven with arbitrary or general interarrival time distribution and Erlang-2 service time distribution

This nomenclature is widely used, and we will adopt it from this point onward.

9.7.1 Additional Reading

[1] D.G.Kendall, *Some Problems in the Theory of Queues,* J.Royal Stat.Society (B), **13,** 151-173, 1951.

[2] P.Kuehn, *Delay Problems in Communications Systems: Classification of Models and Tables for Application,* IEEE International Conference on Communications, **1,** 237-243, Chicago, Illinois, 1977.

9.8 Single Server Queues with Poisson Arrivals and General Service Times

We wish to assess the impact on performance when the arrival statistics are Poisson but the service times for jobs are arbitrary. This allows us to quantify the impact on delay statistics for jobs that have different processing time requirements and also different delay goals: often we wish to execute short jobs that have stringent delay criteria much more quickly than long jobs whose delay criteria are much more loose, and this class of models allows us to quantify the gain due to scheduling to achieve these goals. For example, we might be interested in how variable size packets impact congestion in a packet switching system, so we might choose to fix the *mean* packet length but allow the *distribution* or *variance* to fluctuate to see how performance is affected.

9.8.1 Mean Value Analysis In what follows, we are interested in mean throughput rate and queueing and waiting time statistics. Our mean value analysis allows us to plot the mean throughput rate versus the mean arrival rate, as shown in the figure below.

Figure 9.8.Mean Throughput Rate vs Mean Arrival Rate

As long as the mean arrival rate is less than the maximum rate at which jobs can be serviced, the mean throughput rate equals the mean arrival rate, i.e., the arrival rate is limiting the mean throughput rate. On the other hand, the mean delay can be *anything* from the mean service time on up to infinity, i.e., we can say *nothing* about the mean delay at this level of analysis. Different scheduling policies will lead to different delays.

Once the mean arrival rate exceeds the maximum rate of executing jobs, the single serially reusable resource is a bottleneck. Furthermore, delays will exceed any threshold, because the resource cannot keep up with arrivals, i.e., buffers overflow and so forth.

9.8.2 The M/G/1 Queue with Service in Order of Arrival The benchmark against which we will judge all of our different scheduling policies is the policy of service in order of arrival. We now present one result from the general theory of the *M/G/*1 queueing system. The moment generating function of the waiting time distribution is given by

$$E[exp(-zT_W)] = \frac{z(1-\rho)}{z - \lambda[1 - E[exp(-zT_S)]]}$$

which has mean value given by

$$E(T_W) = \frac{\lambda E(T_S{}^2)}{2(1-\rho)}$$

$E(T_S{}^2)$ is the *second* moment of the service time distribution, or, in other words, the *first* moment of the waiting time distribution depends on *more* than just the first moments of the interarrival and service time distributions. If we adopt the previous definition of squared coefficient of variation, we see

$$E(T_W) = \frac{\lambda E^2(T_S)[1+C_S{}^2]}{2(1-\rho)}$$

$$C_S{}^2 = \frac{variance\ of\ service\ time\ distribution}{(mean\ of\ service\ time\ distribution)^2}$$

A different way of expressing the mean waiting time is

$$E(T_W) = \frac{\rho}{1-\rho} \frac{E[T_S{}^2]}{2E[T_S]}$$

In words, the mean waiting time is the product of two factors, one that depends only on the fraction of time the server is busy, ρ, and one that depends on the fluctuations in the service times for jobs, which is the ratio of the second moment over twice the first moment. Under light loading, $\rho \to 0$, the mean waiting time is negligible, while under heavy loading, $\rho \to 1$, the mean waiting time is inflated or stretched by $1/(1-\rho)$ and can dominate the mean queueing time.

The queueing time (or time in system) moment generating function is given by

$$E[exp(-zT_Q)] = E[exp(-zT_W)]E[exp(-zT_S)]$$

which has mean value

$$E(T_Q) = E(T_W) + E(T_S) = \frac{\lambda E(T_S{}^2)}{2(1-\rho)} + E(T_S)$$

Several special cases are of interest:

- exponential service, $C_S{}^2 = 1$:

$$E(T_W) = \frac{\lambda E^2(T_S)}{1-\rho}$$

- deterministic or constant service, $C_S{}^2 = 0$:

$$E(T_W) = \frac{\lambda E^2(T_S)}{2(1-\rho)}$$

  which is one half the mean waiting time of that of the exponential case

- hyperexponential service, where one might encounter $C_S{}^2 = 100$ for example:

$$E(T_W) = \frac{101 \times \lambda E^2(T_S)}{2(1-\rho)}$$

The random variable N denotes the number of jobs in the system (including the job in execution), and this has moment generating function given by

$$E[X^N] = E[exp(-T_Q[\lambda(1-X)])]$$

>From Little's Law, we recall that the mean number in system equals the mean arrival rate multiplied by the mean time in system:

$$E(N) = \lambda\, E(T_Q)$$

We have summarized these formulae graphically in the following plots of mean queueing time, waiting time, and number in system versus the fraction of time the single server is busy, with the squared coefficient of variation of the service time distribution varied from the deterministic case of zero through the hypoexponential case of one half, through the exponential case of one, and on into the hyperexponential case of one and a half, two, and two and a half. In all cases, the mean service time of a job is one time unit. Two plots are presented, one for utilization varied out to one, and the second for the more typical loading case where the utilization varies up to one half. These plots are useful for assessing transients: if the system is normally loaded to thirty to forty per cent utilization, but suddenly a workload surge raises this to seventy per cent, then the mean values can be see to roughly double or triple in all cases. This suggests doubling system design margins to allow for these longer times and greater amount of storage.

The fraction of time the waiting time and queueing time exceed X is plotted in Figure 9.9, assuming an exponential service time distribution.

This figure shows that under light loading, ρ=0.1, the fraction of time a job waits greater than one service time is under ten per cent, while as the loading increases, $\rho \to 1$, the fraction of time a job waits

Figure 9.9.A.Mean Waiting Time vs Utilization $\rho \leq 1$

Figure 9.9.B.Mean Queueing Time vs Utilization $\rho \leq 1$

ten or more service times becomes larger and larger.

Finally, the random variable T_B denotes the duration of a busy period; the processor is busy, idle, busy, idle, and so on. The moment generating function for the busy period distribution is given implicitly by

$$E[exp(-zT_B)] = E[exp(-T_S(z+\lambda-\lambda E[exp(-zT_B)]))]$$

and hence the mean duration of a busy period is

$$E(T_B) = \frac{E(T_S)}{1-\lambda E(T_S)}$$

9.9 The FS/G/1 Queue with Service in Order of Arrival

What if the arrivals are generated from a finite rather than infinite population? Our model is as follows:

- N identical stations attempt to execute jobs; each station is either idle or is active (either waiting to execute or executing). The idle times for all stations form a sequence of independent identically

Figure 9.9.C.Mean Number in System vs Utilization $\rho \leq 1$

Figure 9.10.A.Mean Waiting Time vs Utilization $\rho \leq 0.5$

distributed exponential random variables with mean idle time $1/\lambda$

- the execution times for each station form a sequence of independent identically distributed random variables, with associated transmission time T_S which has an associated moment generating function $\gamma_{T_S}(z)$ defined by

$$E[exp(-zT_S)] = \gamma_{T_S}(z)$$

- jobs are executed in order of arrival

The mean rate of executing jobs is simply the fraction of time the server is busy executing jobs divided by the mean time to execute one job. We denote by ρ the server utilization, while $E(T_S)$ is the mean job execution time, and see

$$mean\ \ throughput\ \ rate\ = \frac{\rho}{E(T_S)}$$

Figure 9.10.B.Mean Queueing Time vs Utilization $\rho \leq 0.5$

Figure 9.10.C.Mean Number in System vs Utilization $\rho \leq 0.5$

Since each station is either idle or active, the mean cycle time for one station to go from idle to active and back to idle is simply

$$mean\ \ station\ \ cycle\ \ time\ = E(T_{idle}) + E(T_{delay})$$

and by definition the mean throughput rate is the number of stations divided by the mean cycle time per station:

$$mean\ \ throughput\ \ rate\ = \frac{N}{E(T_{idle}) + E(T_{delay})}$$

Equating these two expressions, we see

$$E(T_{delay}) = \frac{N\ E(T_S)}{\rho} - E(T_{idle})$$

The mean utilization is given by

Figure 9.11.A.Fraction of Time $T_Q>X$ versus X

Figure 9.11.B.Fraction of Time $T_W>X$ versus X

$$\rho = \frac{\sum_{J=0}^{N-1} \left\lceil \begin{matrix} N-1 \\ J \end{matrix} \right\rceil \prod_{K=0}^{J} [E[exp(\lambda(K\ T_S))]-1]}{1 + \sum_{J=1}^{N} \left\lceil \begin{matrix} N \\ J \end{matrix} \right\rceil \prod_{K=0}^{J-1} [E[exp(\lambda(K\ T_S))]-1]}$$

Note that we are computing the moment generating function of the job execution time distribution at evenly spaced points. This means that the mean delay depends on *more* than just the mean message length.

EXERCISE: Compute the mean delay versus N for $T_{idle}=1$ and $E(T_S)=0.3$, for exponential and deterministic service time distributions, for a finite source model. Compare these calculations with those from an infinite source model with arrival rate $\lambda=N/T_{idle}$.

Figure 9.11.C.Fraction of Time $N>K$ versus K

9.10 M/G/1 Last Come, First Serve

In this section we examine a different policy for administering a single serially reusable resource: service in reverse order of arrival. Two cases are possible here:

- Upon arrival, the job in service is preempted and the arrival will seize the resource
- Upon arrival, the job in service is not preempted but finishes execution, and the latest arrival then seizes the resource and is completely executed

For the case of preemption, we will only examine the case where service is resumed for the preempted task(s) at the point of interruption. A different case might be to start execution anew or afresh for a preempted job, which we will not deal with here.

Why deal with this policy? Many computer systems employ a hardware device called a *stack* which conceptually is a single serially reusable resource that operates according to a policy of the last arrival is served first.

9.10.1 Nonpreemptive Last Come, First Serve When a job arrives, it will either execute immediately because the system is idle, or it must wait, because one job is in execution. If the job must wait, other jobs may arrive after it but before the first job finished execution, and all of these jobs will be executed before the job of interest is executed. The time in system is given by

$$T_Q = \begin{cases} T_S & \textit{with probability } 1-\rho \\ \tilde{T}_S & \textit{with probability } \rho \end{cases}$$

The moment generating function is given by

$$E[exp(-zT_Q)] = (1-\rho)E[exp(-zT_S)] + \rho E[exp(-z\tilde{T}_S)]$$

If the system is busy upon arrival, a job must wait until the current job in execution is finished, which we denote by $\hat{T}_S$

$$E[exp(-z\hat{T}_S)] = \frac{1-E[exp(-zT_S)]}{zE(T_S)}$$

If we stretch $\hat{T}_S$ to account for arrivals after the job in question, we find $\tilde{T}_S$:

$$E[exp(-z\tilde{T}_S)] = E[exp(-y\hat{T}_S)]$$

$$y = z + \lambda - \lambda E[exp(-zT_B)]$$

$$E[exp(-zT_B)] = E[exp((z + \lambda - \lambda E[exp(-zT_B)]T_S)]$$

The mean queueing time is given by

$$E(T_Q) = E(T_S) + \frac{\lambda E(T_S{}^2)}{2(1-\rho)}$$

This is identical to that for service in order of arrival. The reason is that no preemption is allowed, so when a job completes, another starts (the last one or the first one), and hence the mean number of jobs in the system either waiting to execute or in execution is the same for either policy. Little's Law tells us that the mean queueing time must be the same for the same mean arrival rate.

9.10.2 Preemptive Resume Last Come, First Serve If we allow preemptive resume service, then as soon as a job arrives it begins execution, irrespective of whether the processor or server is busy. However, this job can be preempted by later arrivals. Hence, we see

$$E[exp(-zT_Q)] = E[exp(-yT_S)]$$

$$y = z + \lambda - \lambda E[exp(-zT_B)]$$

$$E[exp(-zT_B)] = E[exp(-(z + \lambda - \lambda E(exp(-zT_B))T_S)]$$

The mean queueing time is given by

$$E(T_Q) = \frac{E(T_S)}{1-\rho}$$

This is the same form as the mean queueing time for an M/M/1 system with service in order of arrival, except that here we have arbitrary service and preemptive resume service in reverse order of arrival. By switching the scheduling policy, the mean value behaves as if the service time were exponentially distributed.

9.10.3 Example The distribution of number of clock cycles per assembly language instruction is measured and found to be adequately modeled by an constant distribution with a mean of two cycles per instruction. We wish to compare the impact on performance using a stack with either nonpreemptive or preemptive resume arbitration versus a fifo or first in, first out buffer discipline. The table below summarizes the mean number of clock cycles per instruction (including both execution and waiting):

Table 9.5.Mean Execution Time $E(T_Q)$

	FIFO	*LCFS*	
Utilization	*Policy*	*Nonpreemptive*	*Preemptive Resume*
0.1	2.1 cycles	2.1 cycles	2.2 cycles
0.5	3.0 cycles	3.0 cycles	4.0 cycles
0.9	11.0 cycles	11.0 cycles	20.0 cycles

The impact due to fluctuations for preemptive resume versus nonpreemptive scheduling is severe as the load grows.

BONUS: What is the variance about the mean for each of these policies?

9.10.4 Graphical Comparisons For the special case of deterministic or constant service time, we can explicitly calculate the exact distribution of T_Q. This is plotted in Figure 9.12 assuming Poisson arrivals with mean arrival rate of one job every two seconds, while the mean service time for each job is one second, so the processor is fifty per cent utilized. In addition, we have plotted the zeroth and first order approximations to the exact solution, plus the modified approximations assuming the fraction of time $T_Q > X$ drops off exponentially as $\exp^{-\alpha X}$. The approximation $g_{0,\alpha}(X)$ is within ten per cent of the exact solution, while $g_{1,\alpha}(X)$ is within five per cent of the exact solution.

By way of comparison, if we change the service time distribution from constant to exponential, and keep everything else the same, then the results are plotted in Figure 9.13. For example, for the constant

Figure 9.12.Last Come First Served Preemptive Resume/Constant Service

Figure 9.13.Last Come First Served Preemptive Resume/Exponential Service

service time distribution case, ninety nine per cent of the jobs are serviced within ten seconds while the corresponding number for exponential service is seventeen seconds.

Finally, what if we vary the policy, but fix the Poisson arrival rate at one job every two seconds, with each job requiring a constant one second of service. Figure 9.14 summarizes numerical approximations:

Ninety per cent of the jobs are executed within four seconds for first in first out service, while this grows to six seconds for nonpreemptive last in first out service, and to seven seconds for preemptive resume last in first out service. The corresponding results for exponential service is shown as follows: Ninety per cent of the jobs are executed within six seconds for first in first out service, while this grows

Figure 9.14.A.Poisson Arrivals/Constant Service/Different Policies

Figure 9.14.B.Poisson Arrivals/Exponential Service/Different Policies

to nine seconds for nonpreemptive last in first out service, and to twelve seconds for preemptive resume last in first out service. Note that this is significantly greater than the constant service case.

9.11 The M/G/1 Queue with Processor Intervisits

As a useful variation on the above example, we consider the following system

- Arrivals obey simple Poisson statistics with mean arrival rate λ
- The service times of jobs are independent identically distributed random variables with T_S denoting the processing time random variable

- Jobs are serviced in order of arrival
- The buffer for work has infinite capacity
- The processor executes all jobs until the system is completely empty of work, and then leaves for a random time interval called an *intervisit* time; the sequence of intervisit times are independent identically distributed random variables with V denoting the intervisit time random variable. If the processor arrives from an intervisit to find the system empty, it immediately leaves to begin another intervisit time.

In many digital systems, the processor is multiplexed amongst a number of jobs, and hence is not available all the time to handle one job type. >From the point of view of any one job, the processor is busy handling its type of work, and then is absent (doing work elsewhere), so this is often a much more realistic model than the model of the previous section (remember: always check *your* particular application to see what assumptions are valid!)..

Let's examine a special case of this to gain insight: $T_S \equiv 0$. This situation is not uncommon in digital systems: often no one job requires a great amount of service, but the processor must handle so many different types of jobs that the time it is absent (doing jobs elsewhere) is much much longer than the time it is present handling any one job type. If the mean time between intervisits is denoted by $E(V)$, then many people would claim that the mean waiting time is simply one half the duration of a mean intervisit time interval, because on the average a job arrives half way through a mean intervisit interval. *This is false!* In fact, the mean waiting time and also the mean queueing time (since the service time is zero) is given by

$$E(T_Q) = E(T_W) + [E(T_S)=0] = \frac{E(V^2)}{2E(V)} = \tfrac{1}{2}E(V)\left[1 + {C_V}^2\right]$$

$${C_V}^2 = \textit{squared coefficient of variation of intervisit} = \frac{var(V)}{E^2(V)}$$

Only if the intervisit times are constant will the mean waiting time be one half of a mean intervisit time: if there are severe fluctuations about the mean, such as with an exponential distribution where the squared coefficient of variation is one, then the mean waiting and queueing time will lengthen (for the exponential distribution the mean waiting and queueing time will be *twice* that of the constant distribution).

This is a very subtle phenomenon: in words, if there is a severe fluctuation about the mean, the impact on congestion will be much worse than might be expected: work will continue to arrive, and the system will take longer and longer to process this work by passing it off to other queues (where it is absent most of the time anyway), compounding the process in a regenerative or positive feedback manner. Some refer to this phenomenon as *length biasing:* arrivals are much more likely to occur during long intervisit time intervals than during short intervisit time intervals, and so on.

How do we show this? Let us denote by t_K the arrival instants of jobs, K=1,2,.... The probability that the waiting time at some time instant say t is less than or equal to some threshold say X is given by

$$PROB[T_W(t) \le X] = \sum_{K=1}^{\infty} PROB[t < t_K \le t + X < t_{K+1}]$$

$$= \sum_{K=1}^{\infty} \int_{t}^{t+X} [1 - G_V(t+X-u)]dG_A[t_K \le u]$$

In words, at least one arrival must occur during $(t,t+X]$ and this event can occur in several mutually exclusive ways, since the last arrival in the time interval $(t,t+X]$ may be the first, the second, the third, and so on. The mean number of arrivals during the time interval $(0,t]$ is equal to

$$E[T_A(0,t)] = \sum_{K=1}^{\infty} PROB[t_K \le t] = \frac{t}{E(V)}$$

Using this, we see

$$PROB[T_W(t) \le X] = \frac{1}{E(V)} \int_t^{t+X} [1 - G_V(t+X-u)]du = \frac{1}{E(V)} \int_0^X [1 - G_V(y)]dy$$

which is independent of t. The moment generating function of this distribution is given by

$$E[exp(-zT_Q)] = \frac{1 - E[exp(-zV)]}{E(V)}$$

If we use the earlier identity, we can determine all moments of integral order.

What about if $T_S \neq 0$? Now we see that the mean waiting time is simply the sum of an intervisit time interval plus the time required to complete the backlog of work that is present when a given arrival occurs, the virtual workload described earlier:

$$T_W = V + W \rightarrow E[exp(-zT_W)] = E[exp(-zV)]E[exp(-zW)]$$

The queueing time is the sum of the waiting time plus the service time:

$$T_Q = T_V + T_W + T_S$$

$$\rightarrow E[exp(-zT_Q)] = E[exp(-zT_V)]E[exp(-zT_W)]E[exp(-zT_S)]$$

The mean waiting time is given by

$$E(T_W) = \frac{E(V^2)}{2E^2(V)} + \left[\frac{\lambda E(T_S^2)}{2(1 - \lambda E(T_S))} = \frac{\lambda E(T_S)}{1 - \lambda E(T_S)} \frac{E[T_S^2]}{2E[T_S]} \right]$$

We have deliberately rewritten the mean waiting time as the product of a term dependent only on mean utilization $\lambda E(T_S) \equiv \rho$ and the randomized mean service time. The mean queueing time is given by

$$E(T_Q) = E(T_W) + E(T_S)$$

As an example of this phenomenon, let's look at the Laplace transform of the waiting time distribution of an M/G/1 queueing system with service in order of arrival:

$$E[exp(-zT_W)] = \frac{(1 - \rho)z}{z + \lambda - \lambda E[exp(-zT_S)]}$$

$$= (1 - \rho) \sum_{K=0}^{\infty} \rho^K E^K[exp(-z\tilde{T}_S)]$$

$$E[exp(-z\tilde{T}_S)] = \frac{1 - E[exp(-zT_S)]}{zE(T_S)}$$

For light loading, $\rho \ll 1$, we see

$$E(T_W) = (1 - \rho) + \rho E(\tilde{T}_S) + \cdots$$

$$E(\tilde{T}_S) = \frac{E(T_S^2)}{2E(T_S)}$$

In words, this says that on the average, under light loading, an arrival will not wait at all a fraction of the time $1 - \rho$, and will wait for one job a fraction of the time ρ with the waiting time being $E(\tilde{T}_S)$, and other terms are negligible (proportional to $\rho^K, K > 1$).

9.11.1 Additional Reading

[1] L.Takacs, **Introduction to the Theory of Queues,** pp.10-11, Oxford University Press, New York, 1962.

9.12 Synchronous Data Link Control

A widely used data link control procedure is called *Synchronous Data Link Control (SDLC).* and its international standard cousin (HDLC). Compared with previous widely used data link control procedures, it offers a number of advantages:

- It is insensitive to the actual character set being used, because it deals with bit streams, and knows nothing about character sets
- The encoding and buffering is simplified because it is done on the fly, as bits arrive
- A very high link efficiency is achievable compared with other widely used approaches

This does not come for free. Its' disadvantages include

- Variable lengths of messages or frames, which leads to complicated buffering strategies compared to earlier approaches
- The overhead is dependent on the pattern of ones and zeros in the data
- Certain types of single bit errors are undetectable; how often these errors occur can determine how suitable this is in a given application

How does SDLC function? Data arrives at a link controller, is encoded with appropriate address, control field, and error detecting and/or correcting cylic redundancy checking (CRC) coding, and then transmitted over a link. Each frame begins and ends with a unique bit pattern called a *flag* that delimits frames from one another.

Figure 9.15.SDLC Frame Format

Contention for the link is arbitrated using a nonpreemptive priority policy: data has highest priority, while at the lower priority level a flag is always present to be sent. If there is no data, a flag is transmitted. At the end of each flag transmission, the controller checks to see if any data is ready to be transmitted, and if so, begins transmission; otherwise, a flag is transmitted, and the process repeats itself.

Figure 9.16.Priority Arbitration Queueing Network Block Diagram

Flags delimit the start and finish of data transmission: a flag is inserted at the beginning and appended to the end of every data transmission. The flag consists of R+2 bits: a leading zero bit, R+1 successive one bits, and a trailing zero bit. It is mandatory that the flag pattern *not* appear in the middle of the data. Special circuitry in the link encoder monitors the total data transmission: if a zero is followed by R successive ones, the special circuitry at the transmitter inserts a zero immediately after the R successive ones, i.e., it stuffs a bit into the data, and hence is called a *bit stuffer*. At the receiver all stuffed bits are removed by an analogous process. Figure 9.17 illustrates a thirteen bit data stream with

a four bit flag (a leading zero, two ones, and a zero) that leads to an encoded stream of nineteen bits: two flags of four bits each, thirteen data bits, and two inserted or stuffed bits.

Figure 9.17.Illustrative Bit Stuffing Example: D=13, R=2

In the following example, a single bit channel error in bit position five results in the received frame being a flag, which would be undetected by the controller described here (but presumably would be caught by other protocol levels).

Figure 9.18.Illustrative Spurious Flag from Single Bit Error

There are D data bits per message, with a mean of $E(D)$ data bits and a variance of $var(D)$ $bits^2$ per message. There are A=8 address bits, C=8 control bits, and CRC=16 error coding bits associated with each message. The total number of bits to be encoding by bit stuffing is $B=D+32$ bits. Figure 9.19 summarizes both the frame encoding, bit stuffing encoding, and link arbitration.

If P and $Q=1-P$ denote the fraction of D bits that are ones and zeros, respectively, then the mean and variance of the number of data bits between inserted or stuffed bits is

$$mean\ number\ of\ bits\ between\ bit\ stuffs \approx \frac{B}{\mu}$$

Figure 9.19.SDLC Link Controller Block Diagram

$$variance\ of\ number\ of\ bits\ between\ bit\ stuffs \approx \frac{B}{\mu}\ \frac{\sigma^2}{\mu^2} + \left[\frac{B}{\mu}\right]^2 var(D)$$

$$\mu = \frac{1 - P^R}{Q\ P^R} \qquad \sigma^2 = \frac{1}{(Q\ P^2)^2} - \frac{2R+1}{q\ P^R} - \frac{P}{Q^2}$$

SDLC employs a one byte flag: R=5 with the flag consisting of a zero, six ones, and a zero. For P=Q=½, i.e., equal fractions of ones and zeros, the mean number of bits that are transmitted before a bit stuffing occurs is sixty two bits, μ=62 *bits*. Put differently, the mean overhead is one bit out of every sixty three bits, or 1.5873 %. For P=2/3, i.e., two thirds of the bits are ones, one bit is stuffed on the average every 19.78 bits, for a bit stuffing overhead of 4.812%; for P=1/3, i.e., one third of the bits are ones, one bit is stuffed on the average every 363 bits, resulting in bit stuffing overhead of 0.274%. This makes it clear that the bit stuffing overhead incurred by this type of encoding uses relatively few bits to achieve a *transparent* encoding of a bit stream: note that there are no special control characters with this encoding strategy, unlike other approaches such as *Binary Synchronous Communications.* Furthermore, the overhead is relatively insensitive to the proportion of ones and zeros in a frame.

BONUS: What is the overhead per frame for a four bit flag, R=1 as a function of $0<P<1$? Repeat for a two byte flag, R=13.

A *frame* consists of a leading flag, an address field, a control field, a data field, a CRC field, and a trailing flag. The mean number of bits per frame is given by

$$mean\ number\ transmitted\ bits/frame\ = 2(R+3) + B\ \left[1 + \frac{1}{\mu}\right]$$

The first term is due to the two flags, the term B is due to the control and data bits, and the final term B/μ is due to bit insertions among the control and data bits. The variance of the number of bits in a frame is given by

$$variance(number\ bits\ transmitted/frame) = \sigma_{frame}{}^2 = \frac{B}{\mu}\ \frac{\sigma^2}{\mu^2} + \frac{var(D)}{\mu^2}$$

The maximum rate at which the link can transmit the frame is given by

$$\lambda_{\max} = \frac{link\ data\ rate\ (bits/sec)}{mean\ number\ bits/frame}$$

The table below summarizes the mean and variance of the number of bits per frame transmitted (including control bits and inserted or stuffed bits) assuming equal fractions of ones and zeroes in the data bits, with parameters being the mean number of data bits per frame, and the squared coefficient of the number of data bits per frame (denoted $C_{data}{}^2$):

Table 9.6.Bits Transmitted/Frame Statistics(Rounded up to Nearest Integer)

Mean Data Bits/Frame	*Mean Bits Transmitted/Frame*	σ_{frame} $C_{data}{}^2=0$	$C_{data}{}^2=1$
500	589	3	9
1000	1097	5	13
2000	2113	6	33

This numerical summary shows that this encoding method incurs a small amount of overhead beyond the needed seventy two bits per frame for flags, address and control fields, and CRC coding, and that the fluctuations about the mean measured in units of standard deviations are relatively modest.

If the message arrival statistics to the controller can be adequately modelled by a Poisson process with rate λ, then the mean delay, including bit stuffing and transmission, from when the first bit of the message arrives until the last bit is transmitted is the sum of three terms:

- a term accounting for the delay in waiting for the last flag to be transmitted, and since each flag consists of a fixed number of $R+3$ bits each frame will be delayed on the average by $\frac{1}{2}(R+3)$ bit transmission times
- a term accounting for the delay while the backlog due to previously arrived frames is transmitted
- a term due to transmitting the frame

Combining all this, we find:

$$mean\ \ frame\ \ delay\ \ = E[T_Q] = E[T_{frame}] + \tfrac{1}{2}T_{flag}$$

$$+ \begin{cases} \dfrac{\lambda E[T_{frame}]}{1-\lambda E[T_{frame}]}\ \dfrac{variance\,(number\ \ of\ bits/frame}{2(mean\ \ number\ \ of\ bits/frame)^2} & \lambda<\lambda_{\max} \\ \infty & \lambda\geq\lambda_{\max} \end{cases}$$

$$E[T_{frame}] = \frac{mean\ \ number\ \ of\ bits/frame}{link\ \ data\ \ rate\,(bits/sec)} \qquad T_{flag} = \frac{R+3\ bits/flag}{link\ \ data\ \ rate\,(bits/sec)}$$

EXERCISE: Plot the mean frame delay versus message arrival rate for 500, 1000, and 2000 data bits per frame with a four, eight, and sixteen bit flag, assuming $P=\frac{1}{2}$ and $P=2/3$.

9.12.1 Additional Reading

[1] R.J.Camrass, R.G.Gallager, *Encoding Message Lengths for Data Transmission*, IEEE Transactions on Information Theory, **24** (4), 495-496 (1978).

[2] R.L.Donnan, R.Kersey, *Synchronous Data Link Control: A Perspective*, IBM Systems Journal, **13** (2), 140-161, 1974.

[3] J.S.Ma, *On the Impact of HDLC Zero Insertion and Deletion on Link Utilization and Reliability*, IEEE Transactions on Communications, **30** (2), 375-381 (1982).

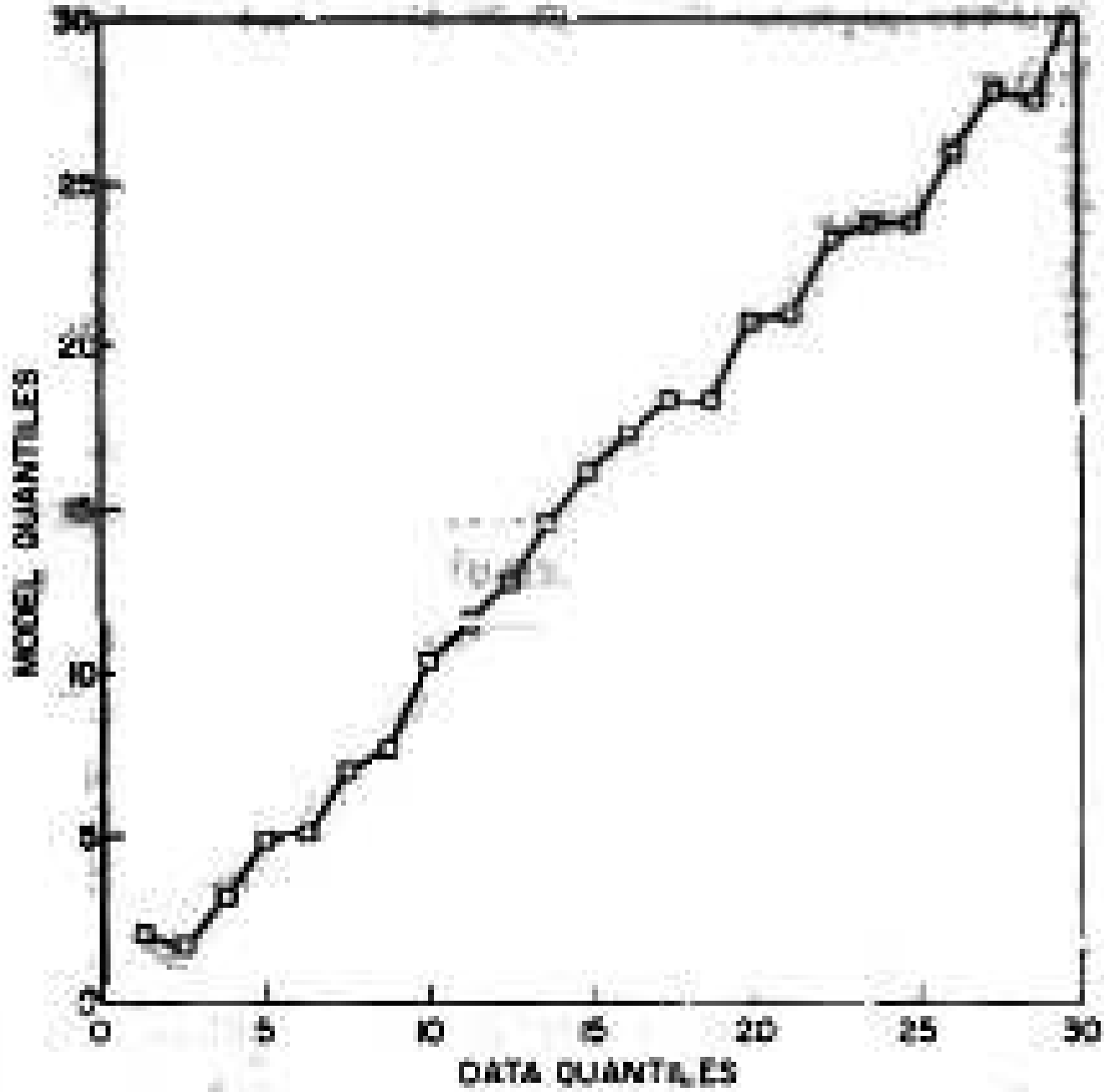

Figure 9.1.Empirical Quantiles vs Exponential Model Quantiles

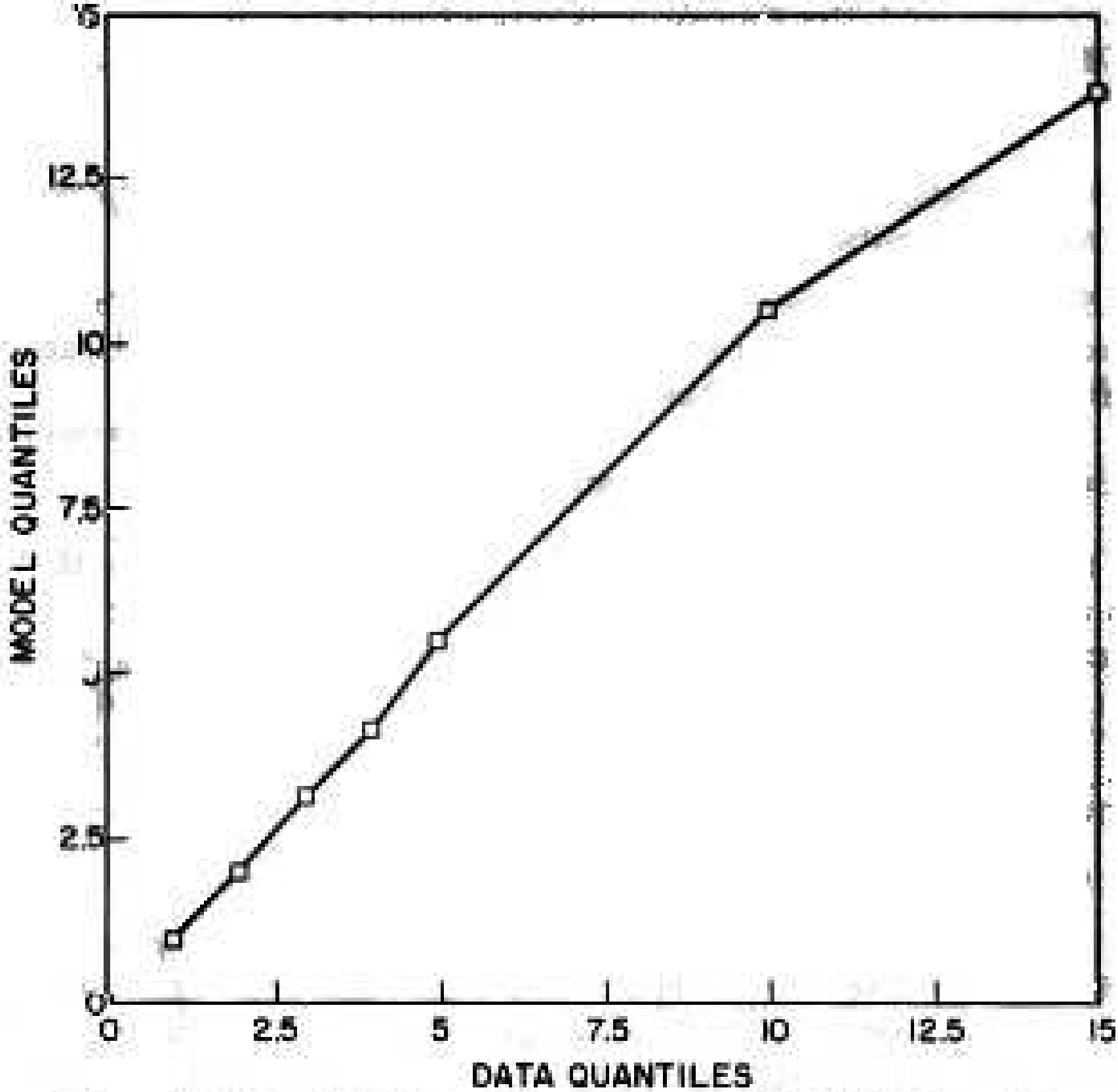

Figure 9.2.Empirical Quantiles vs Exponential Model Quantiles

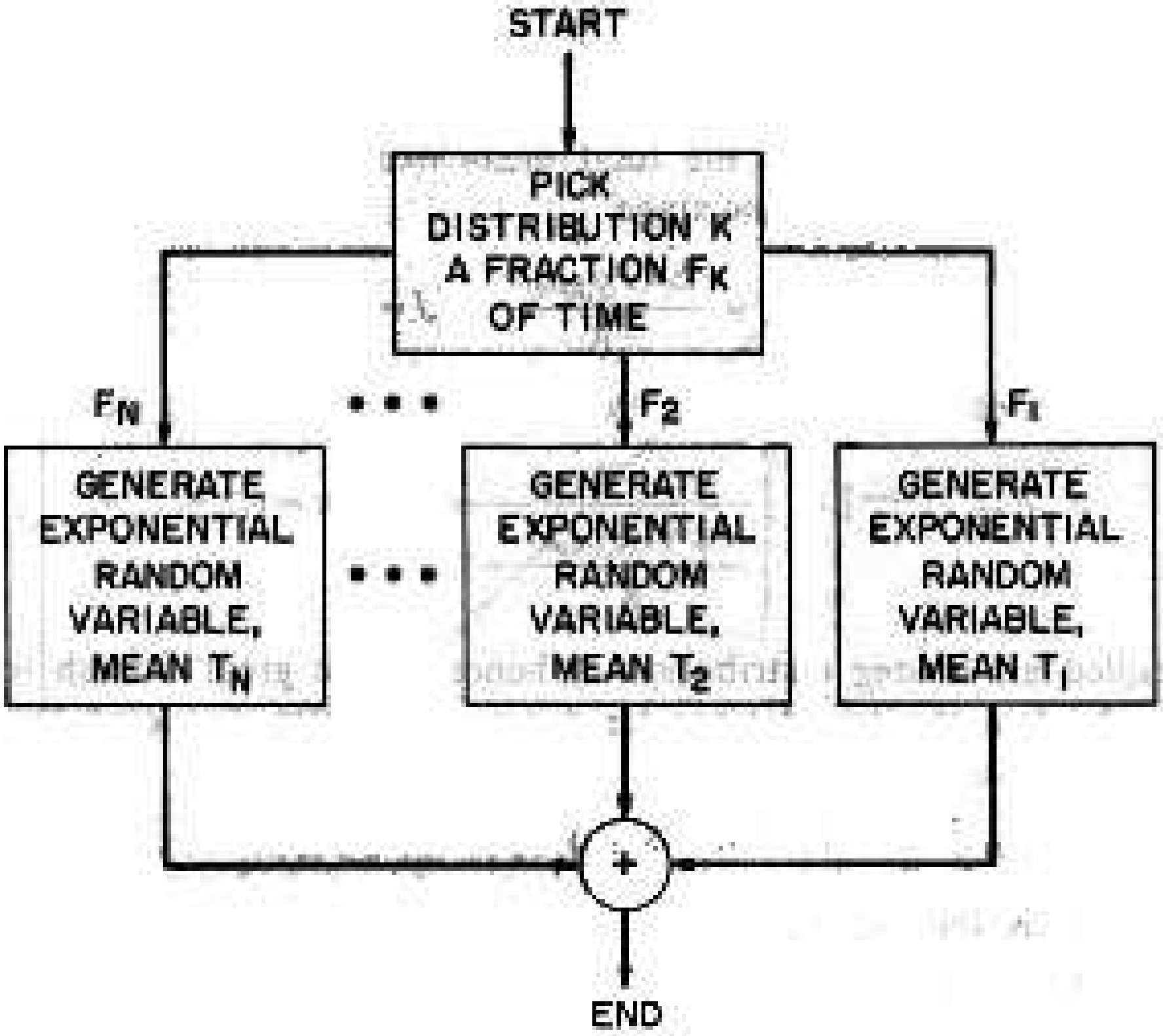

Figure 9.3.Hyperexponential Random Variable Generation

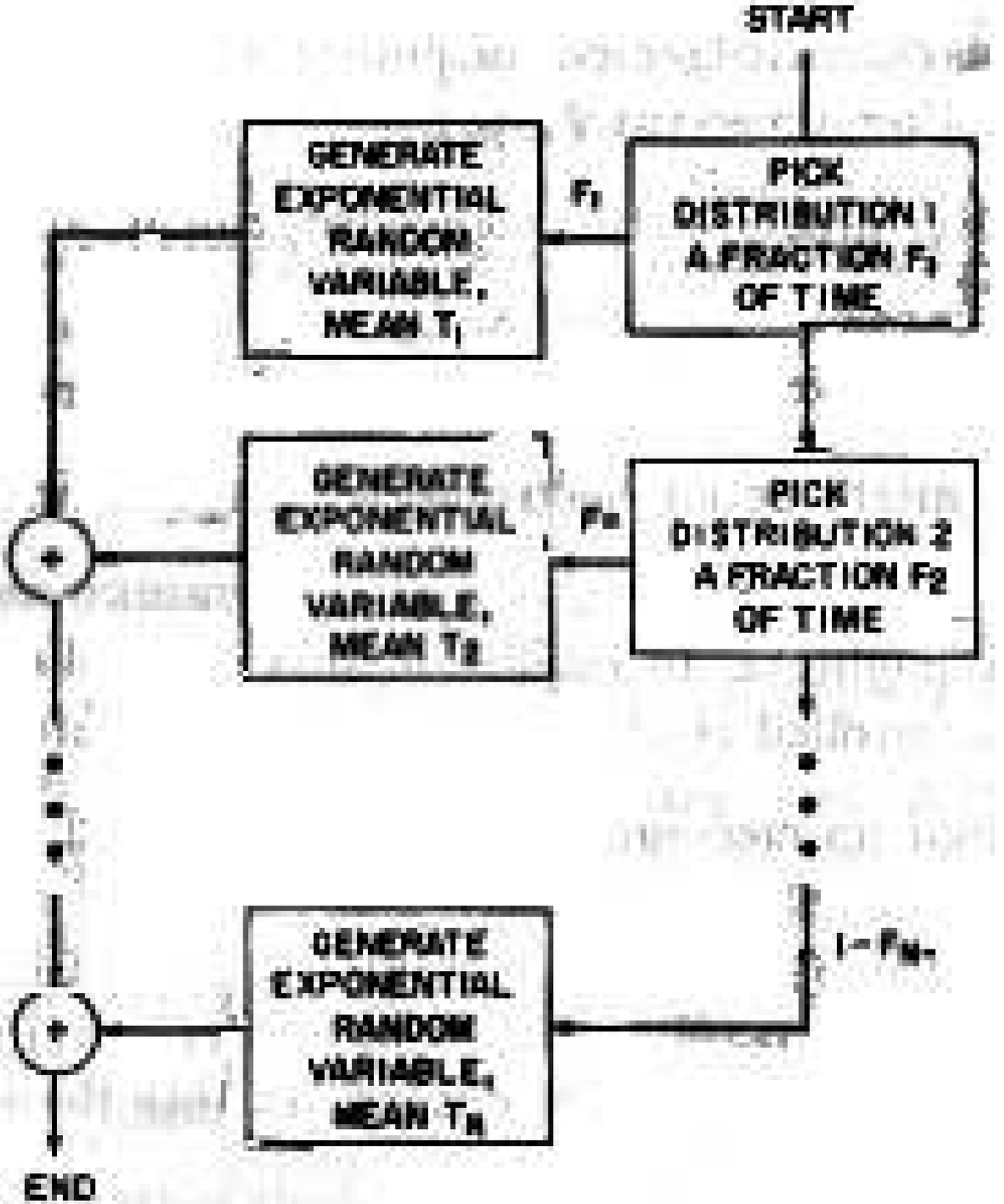

Figure 9.4.Hypoexponential Distribution Flow Chart

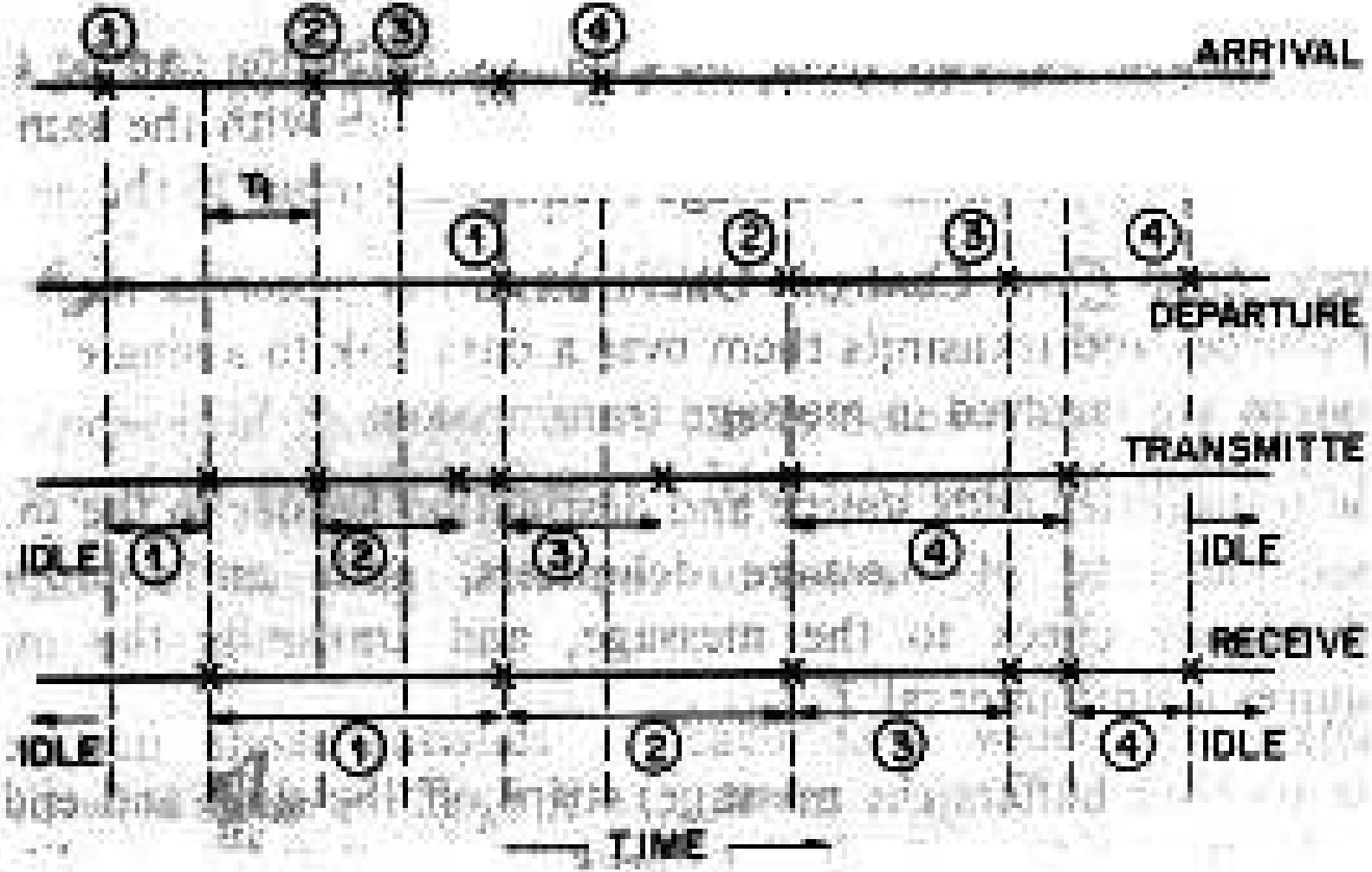

Figure 9.5.Illustrative Operation with Negligible Propagation Time

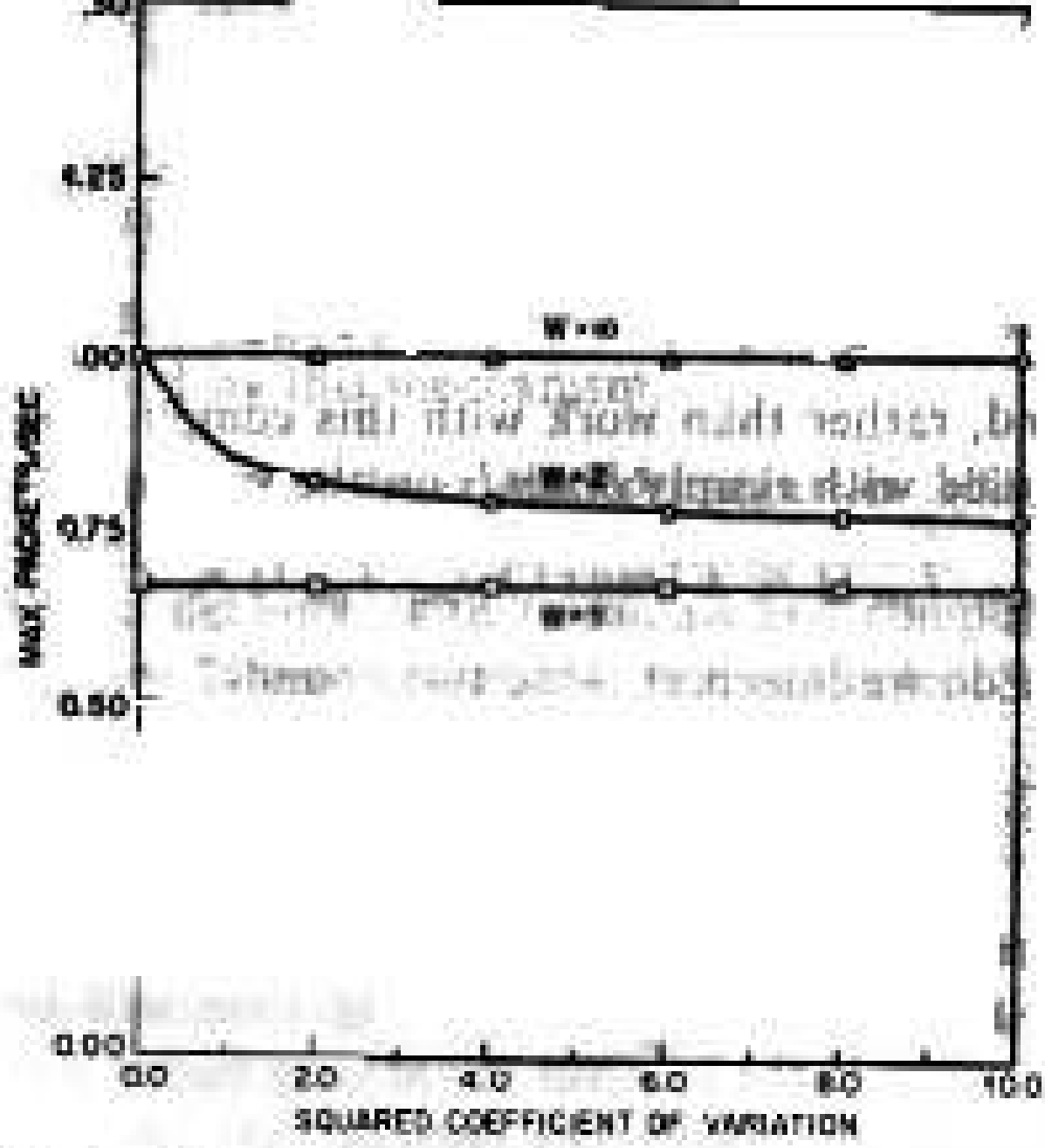

Figure 9.6.Mean Throughput Gain of Double vs Single Buffering

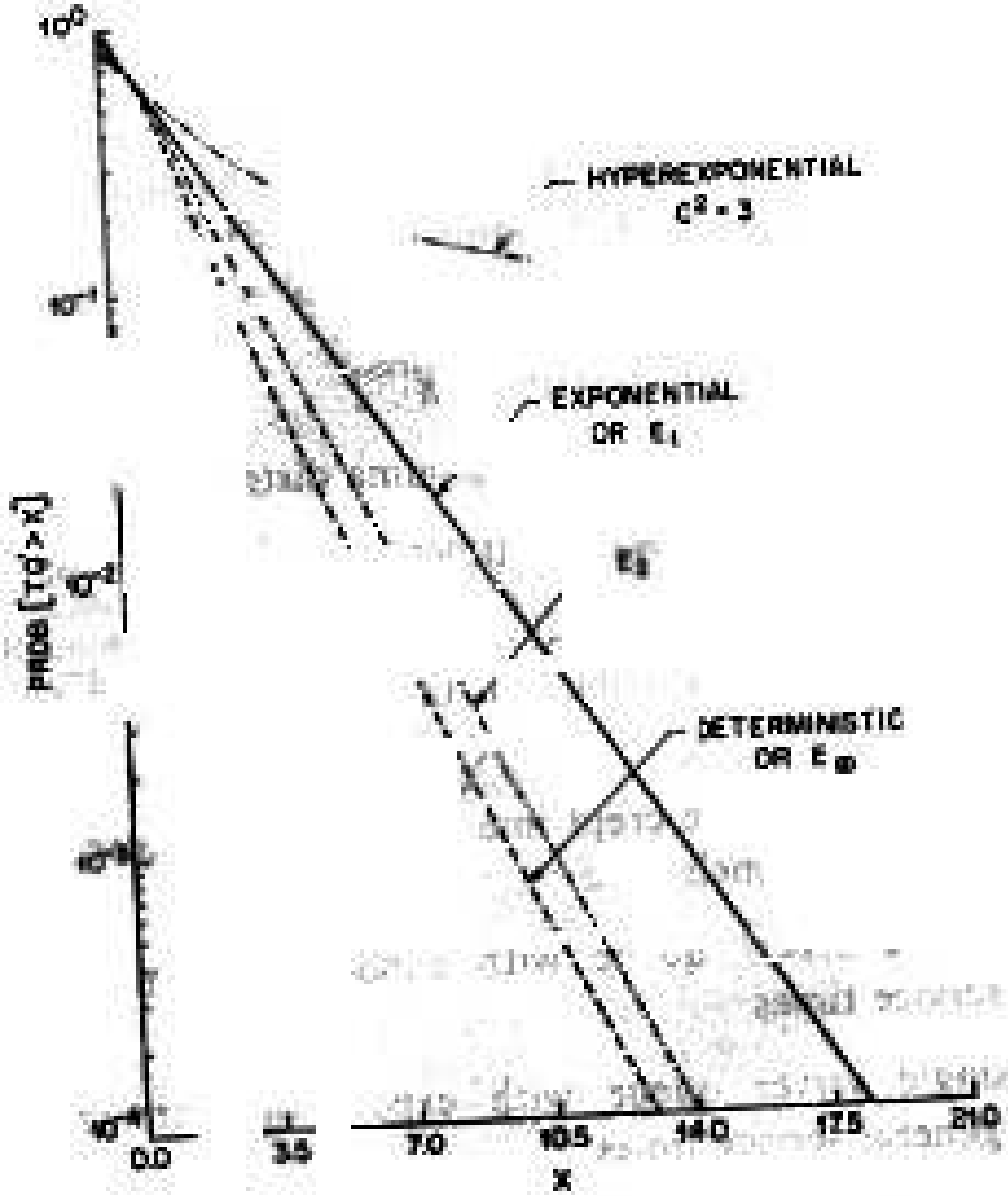

Figure 9.7.Fraction of Time $T_Q > X$ vs X

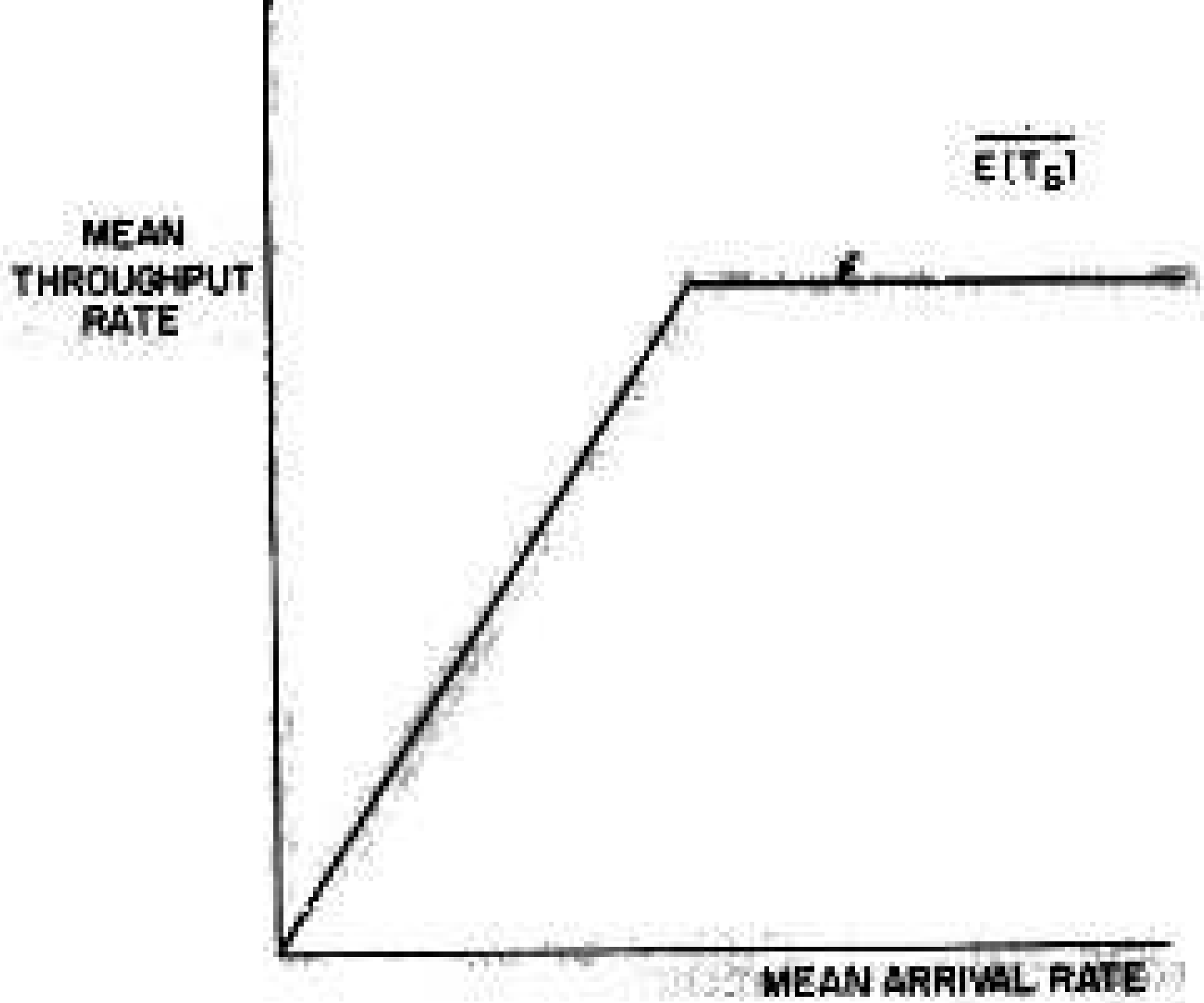

Figure 9.8.Mean Throughput Rate vs Mean Arrival Rate

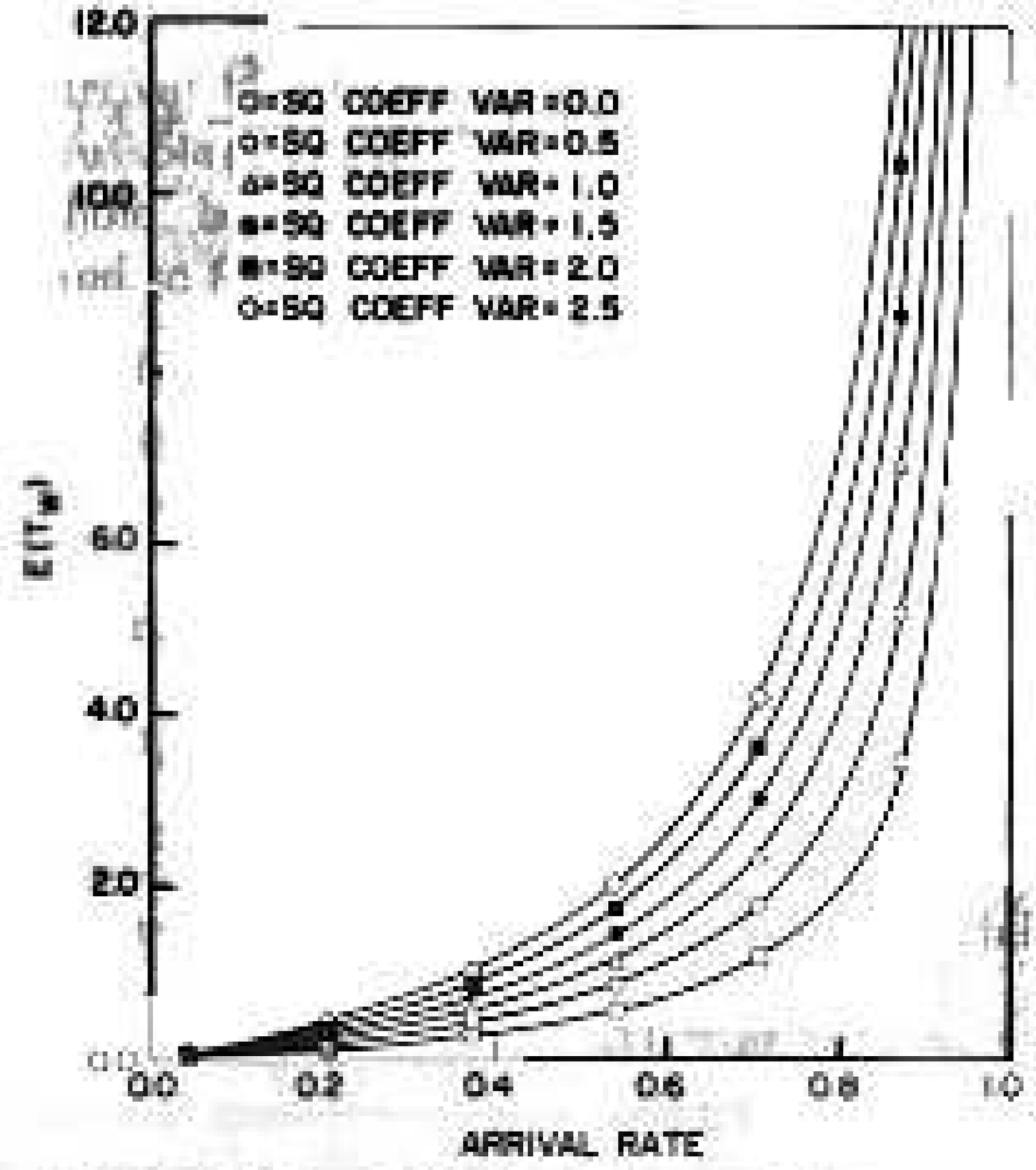

Figure 9.9.A.Mean Waiting Time vs Utilization $\rho \leq 1$

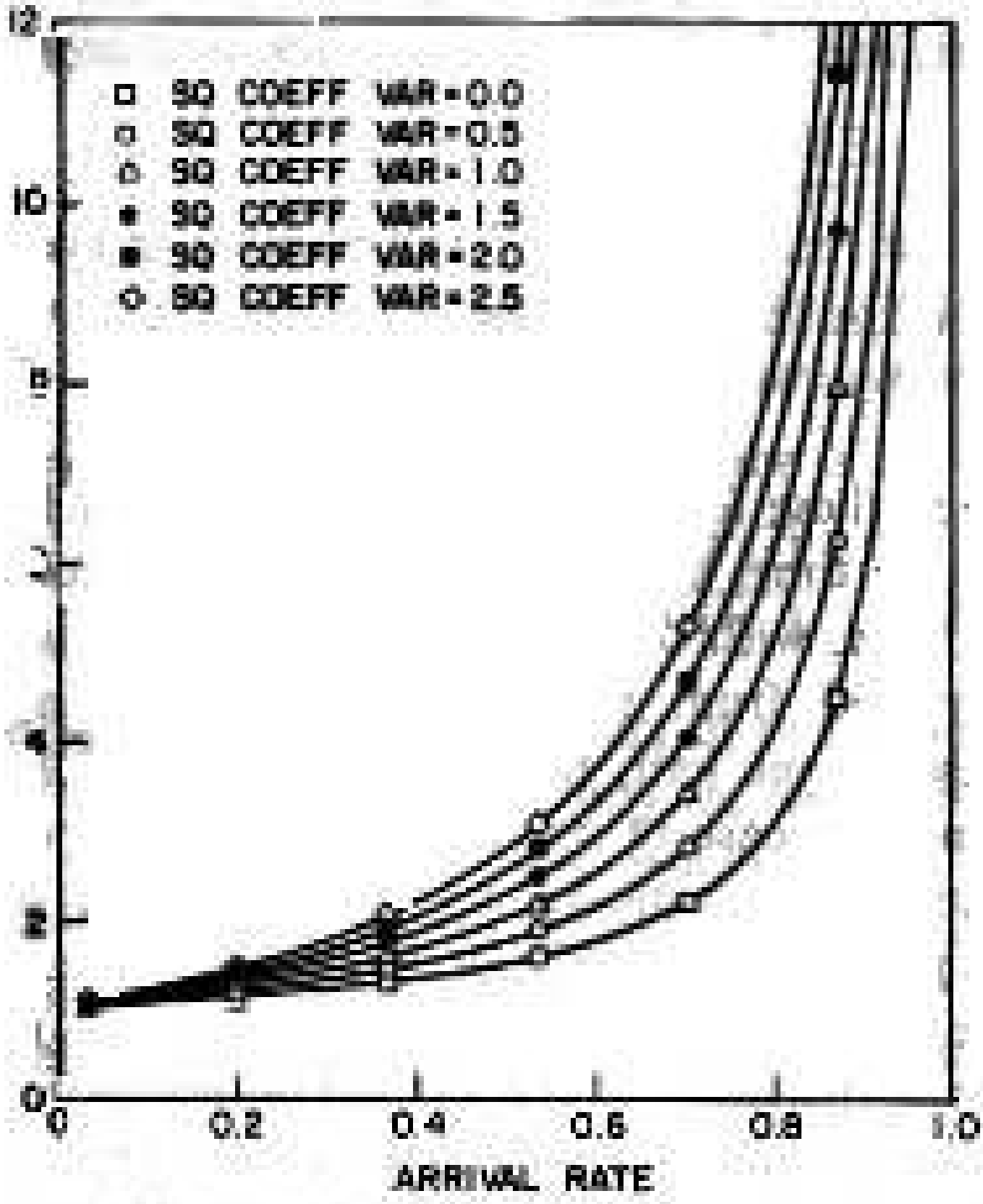

Figure 9.9.B.Mean Queueing Time vs Utilization $\rho \leq 1$

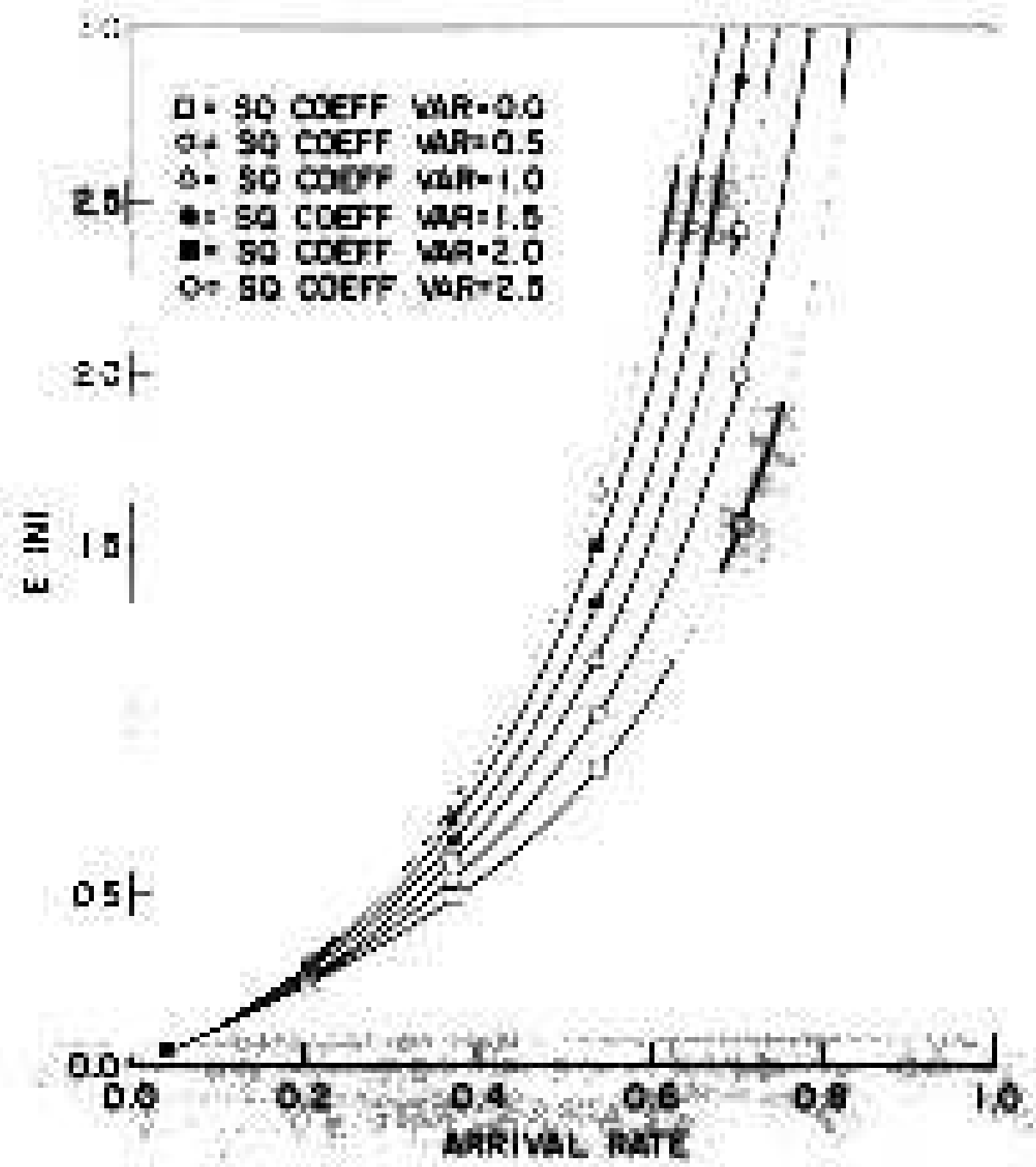

Figure 9.9.C.Mean Number in System vs Utilization $\rho \leq 1$

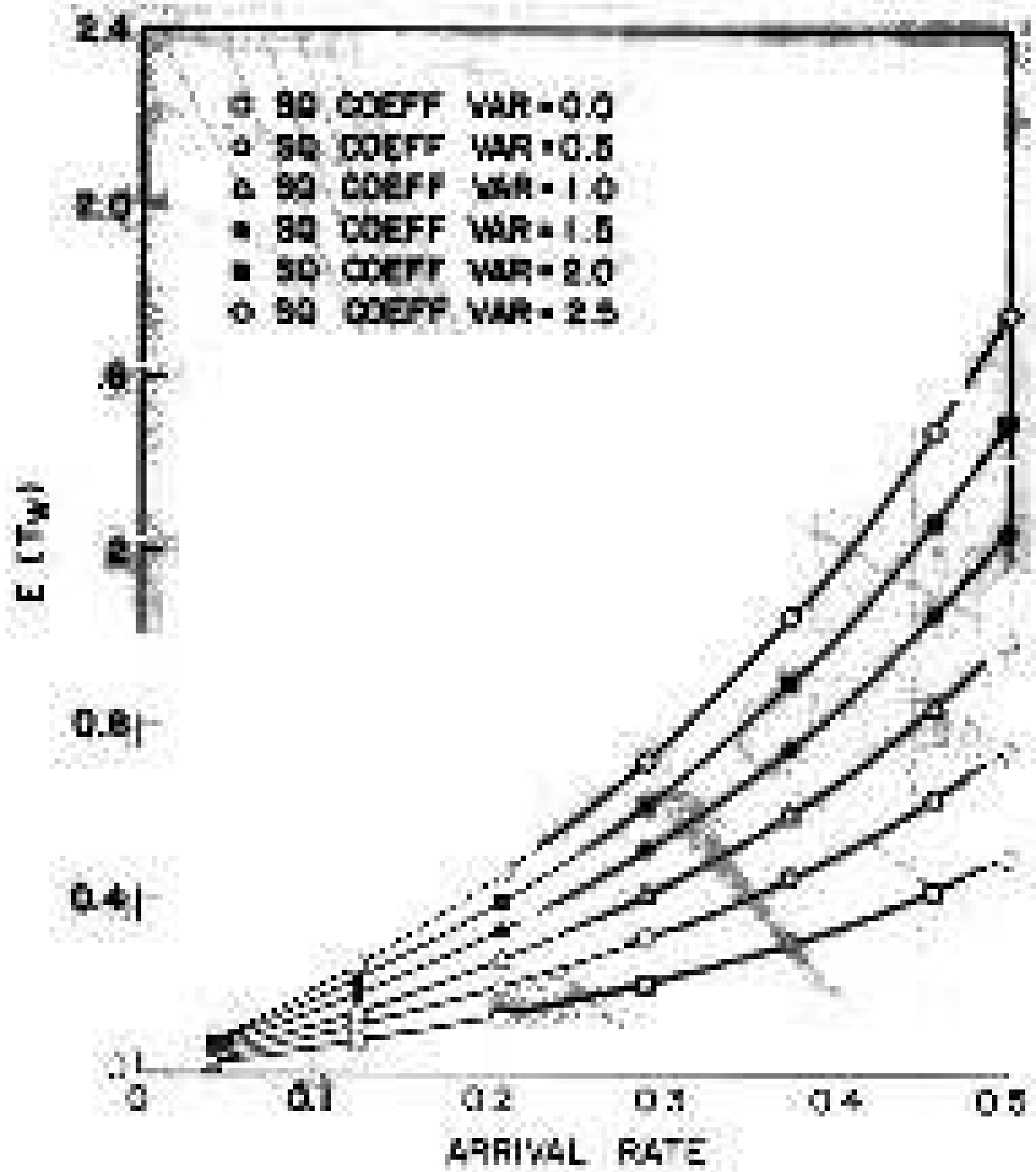

Figure 9.10.A.Mean Waiting Time vs Utilization $\rho \leq 0.5$

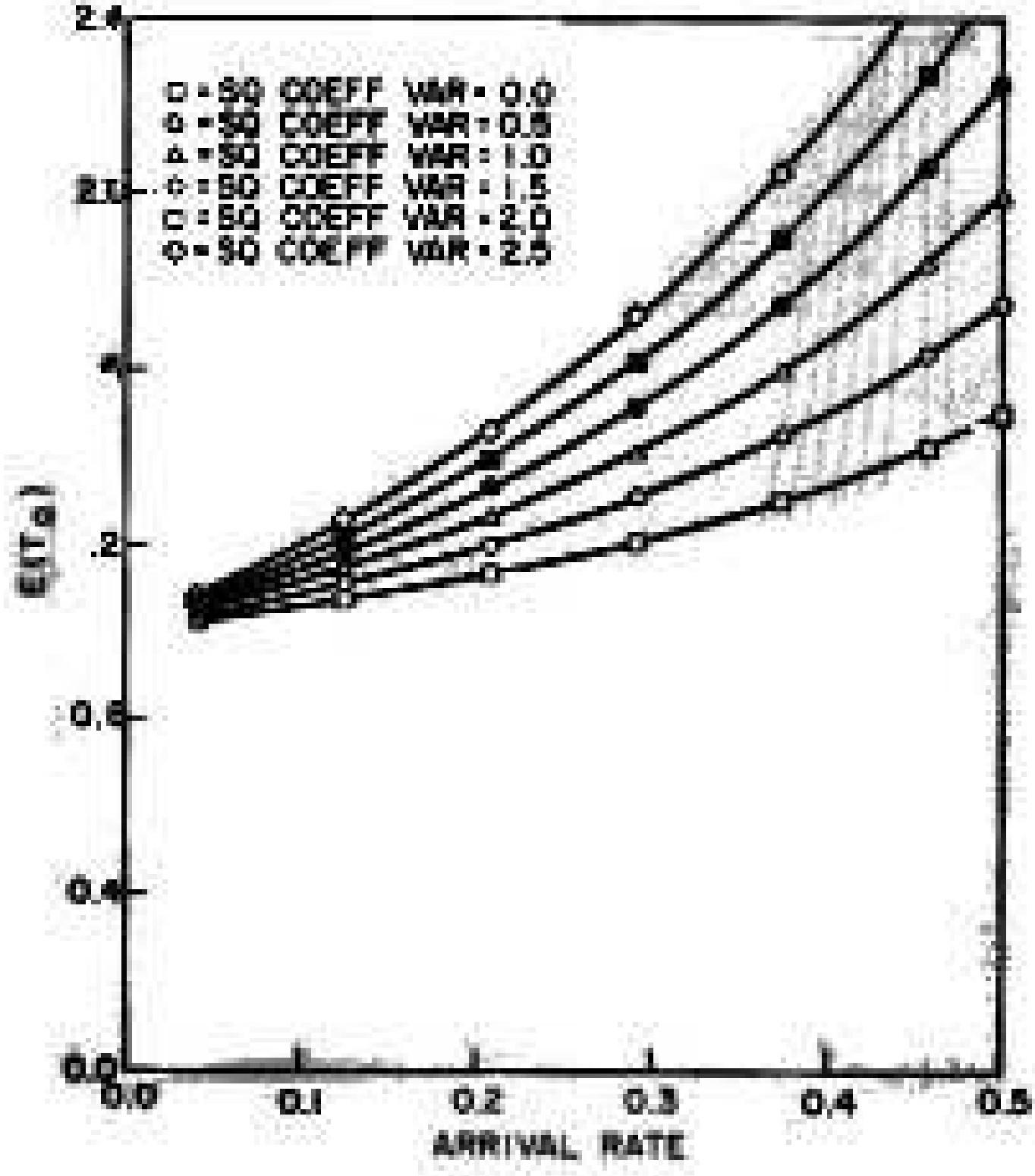

Figure 9.10.B.Mean Queueing Time vs Utilization $\rho \leq 0.5$

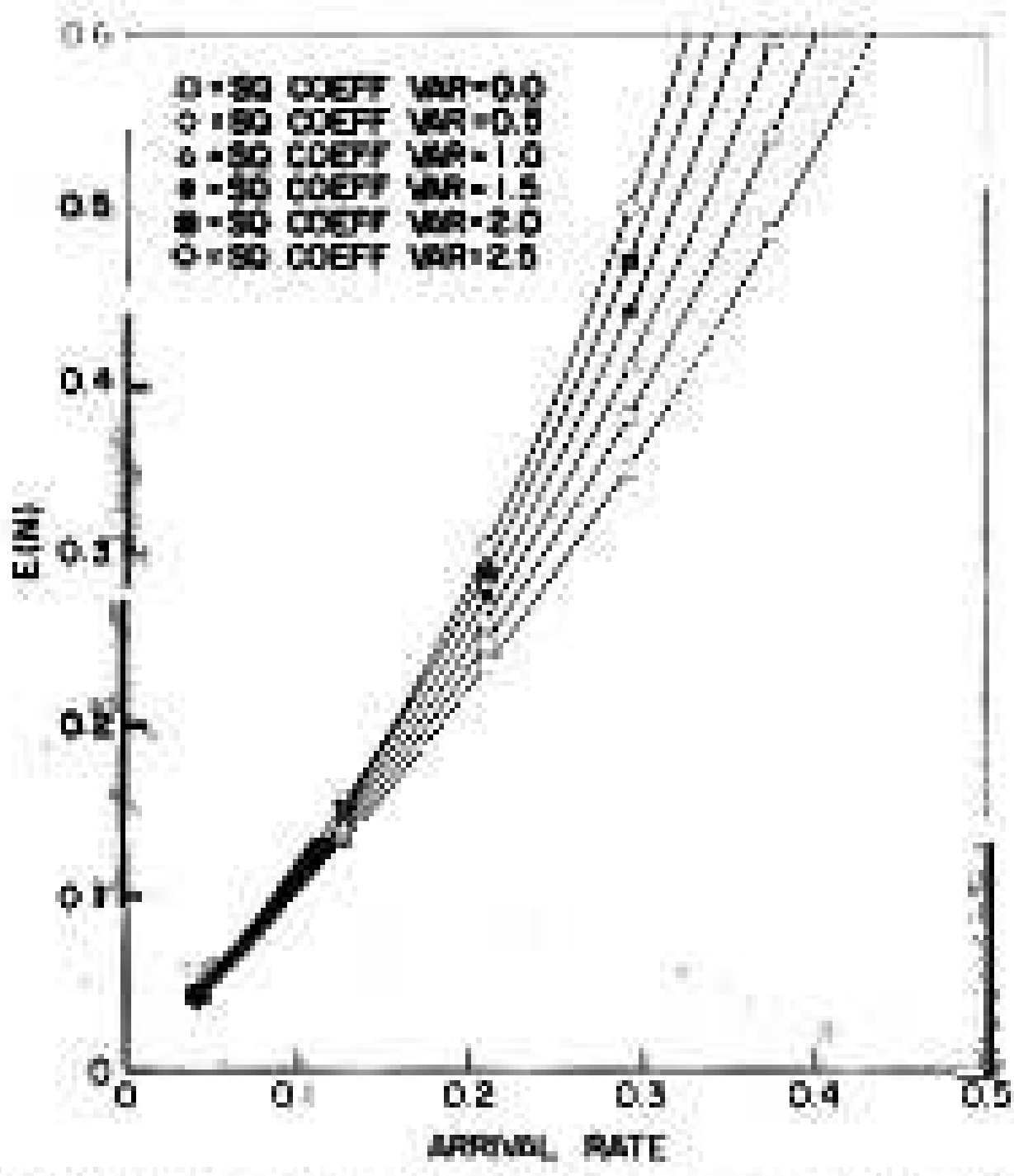

Figure 9.10.C.Mean Number in System vs Utilization $\rho \leq 0.5$

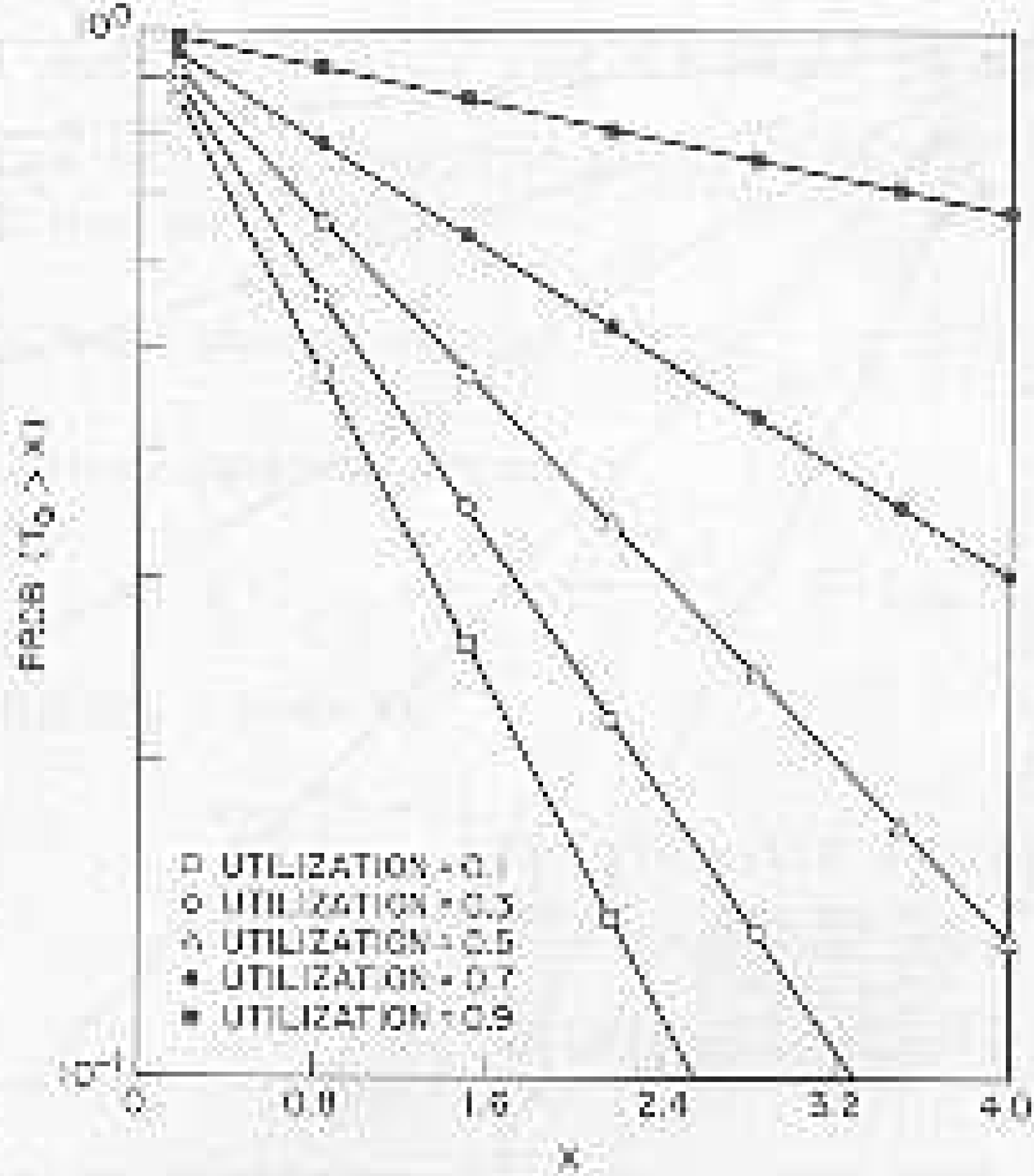

Figure 9.11.A.Fraction of Time $T_Q > X$ versus X

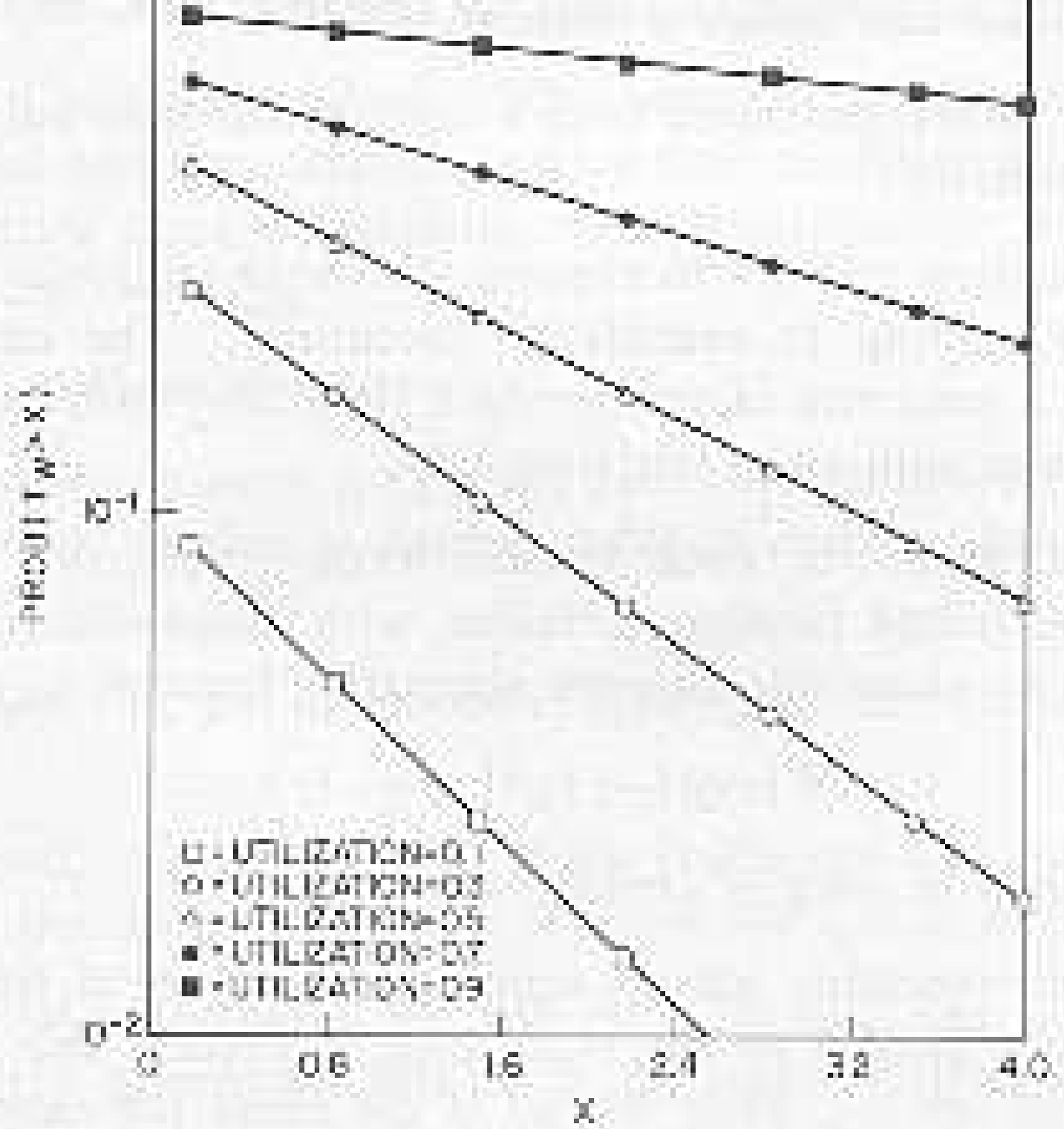

Figure 9.11.B. Fraction of Time $T_W > X$ versus X

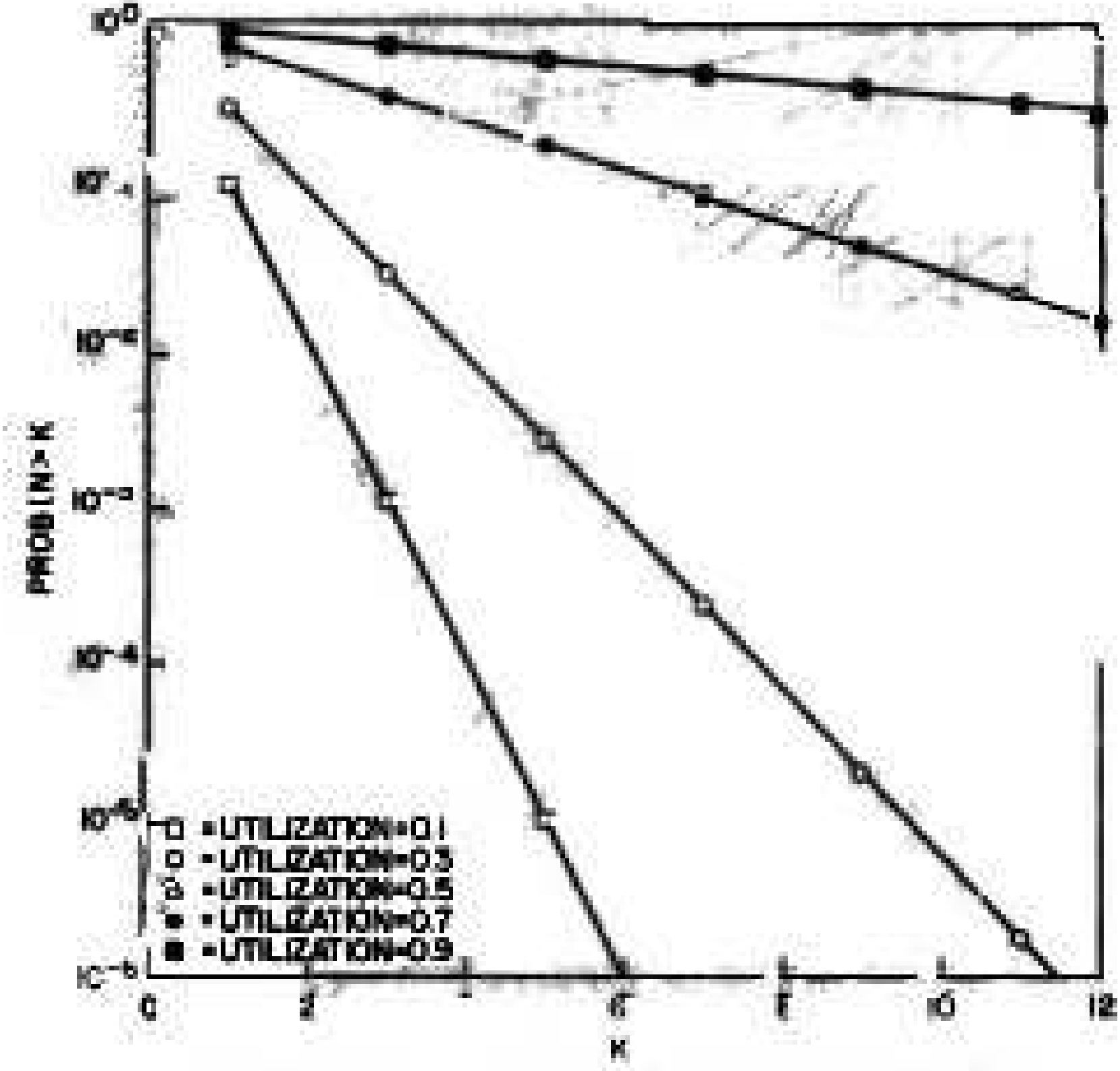

Figure 9.11.C.Fraction of Time $N > K$ versus K

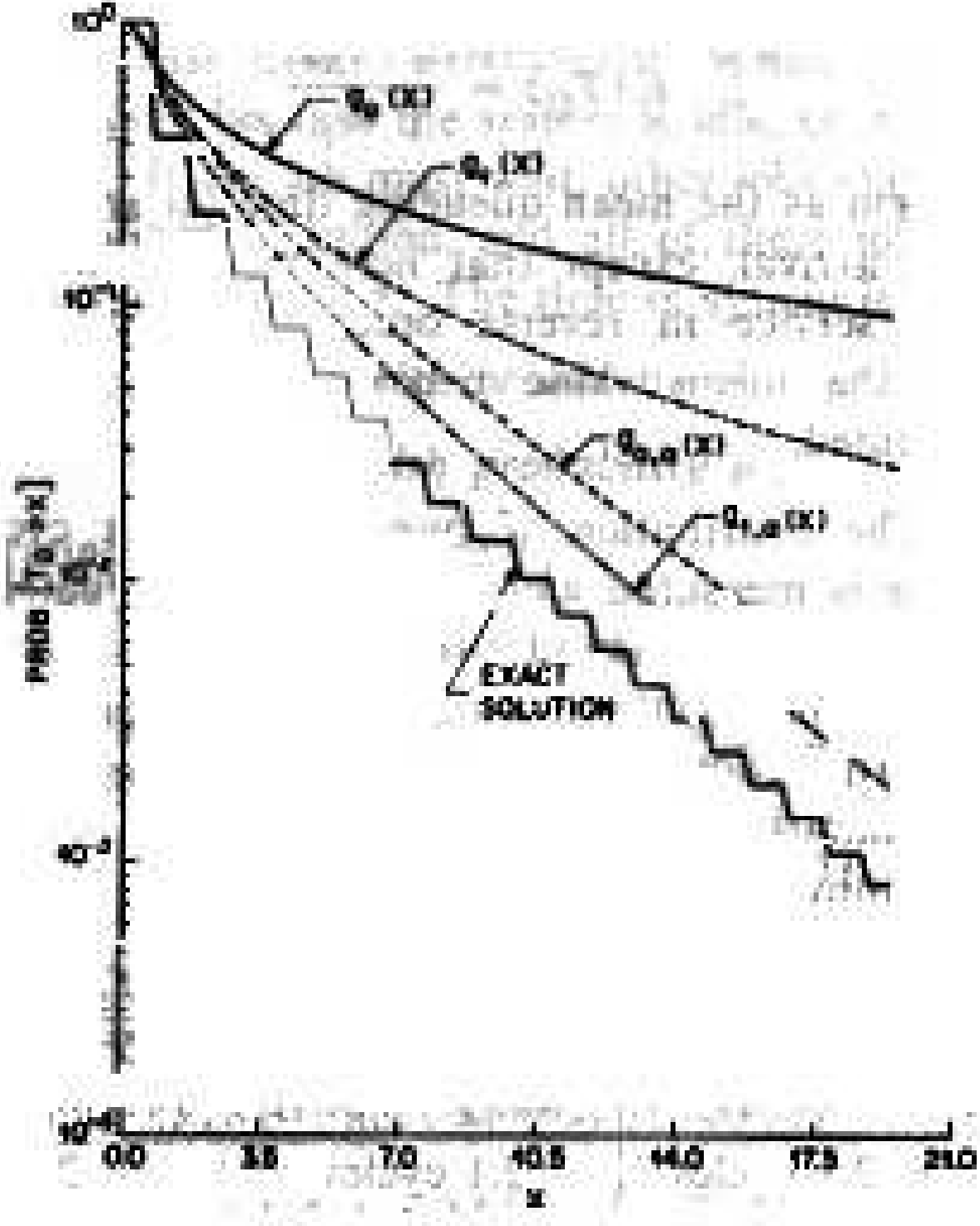

Figure 9.12.Last Come First Served Preemptive Resume/Constant Service

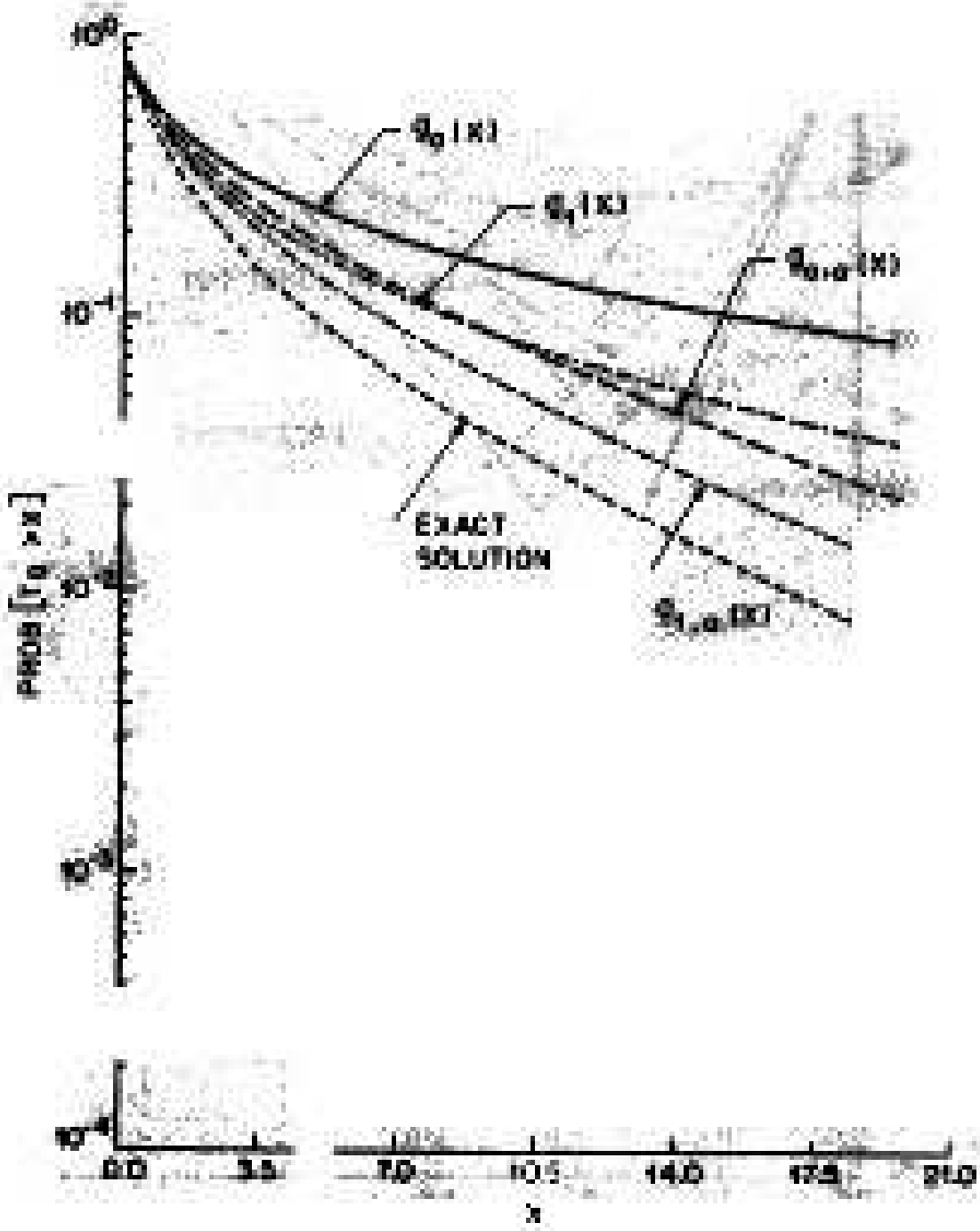

Figure 9.13.Last Come First Served Preemptive Resume/Exponential Service

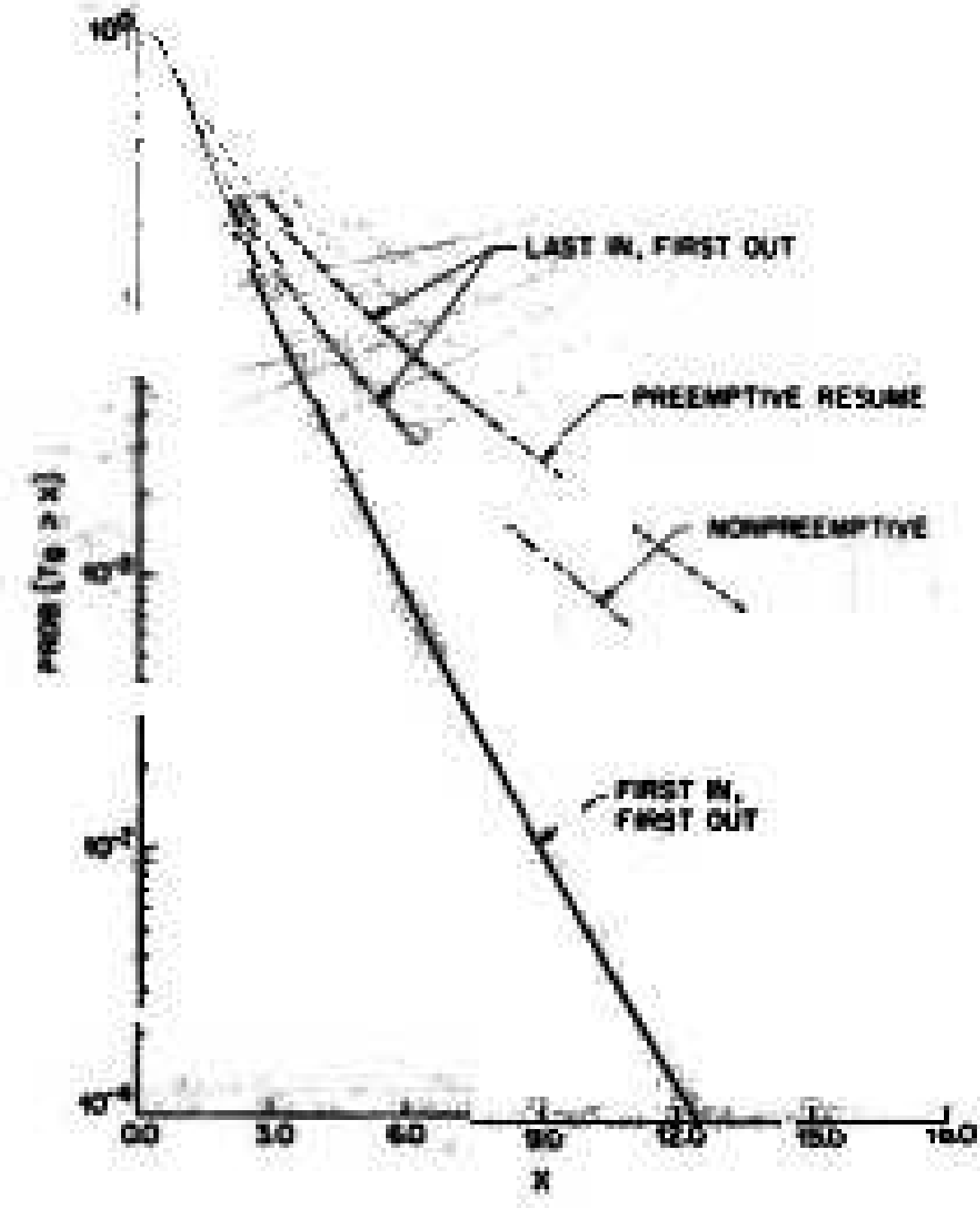

Figure 9.14.A.Poisson Arrivals/Constant Service/Different Policies

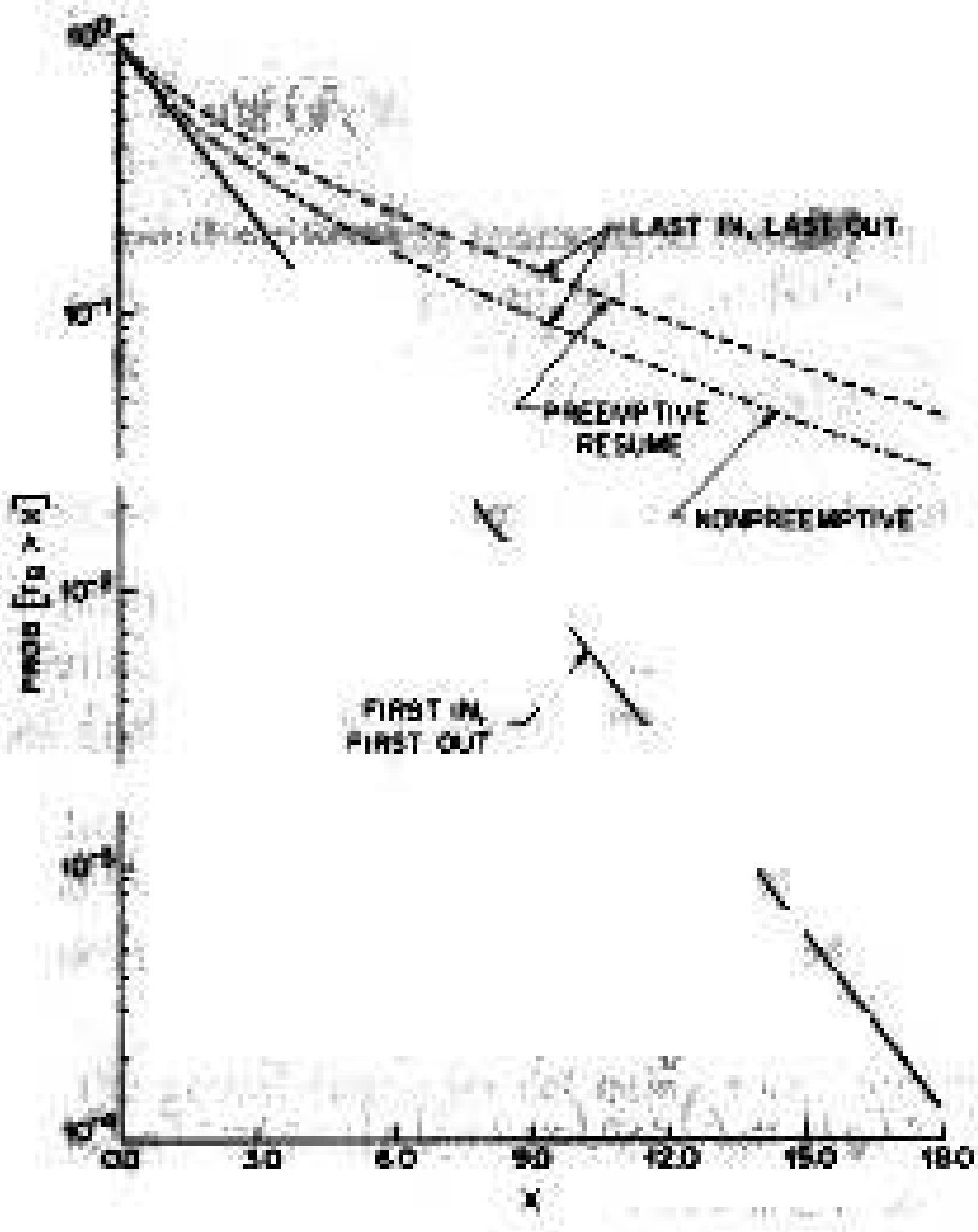

Figure 9.14.B.Poisson Arrivals/Exponential Service/Different Policies

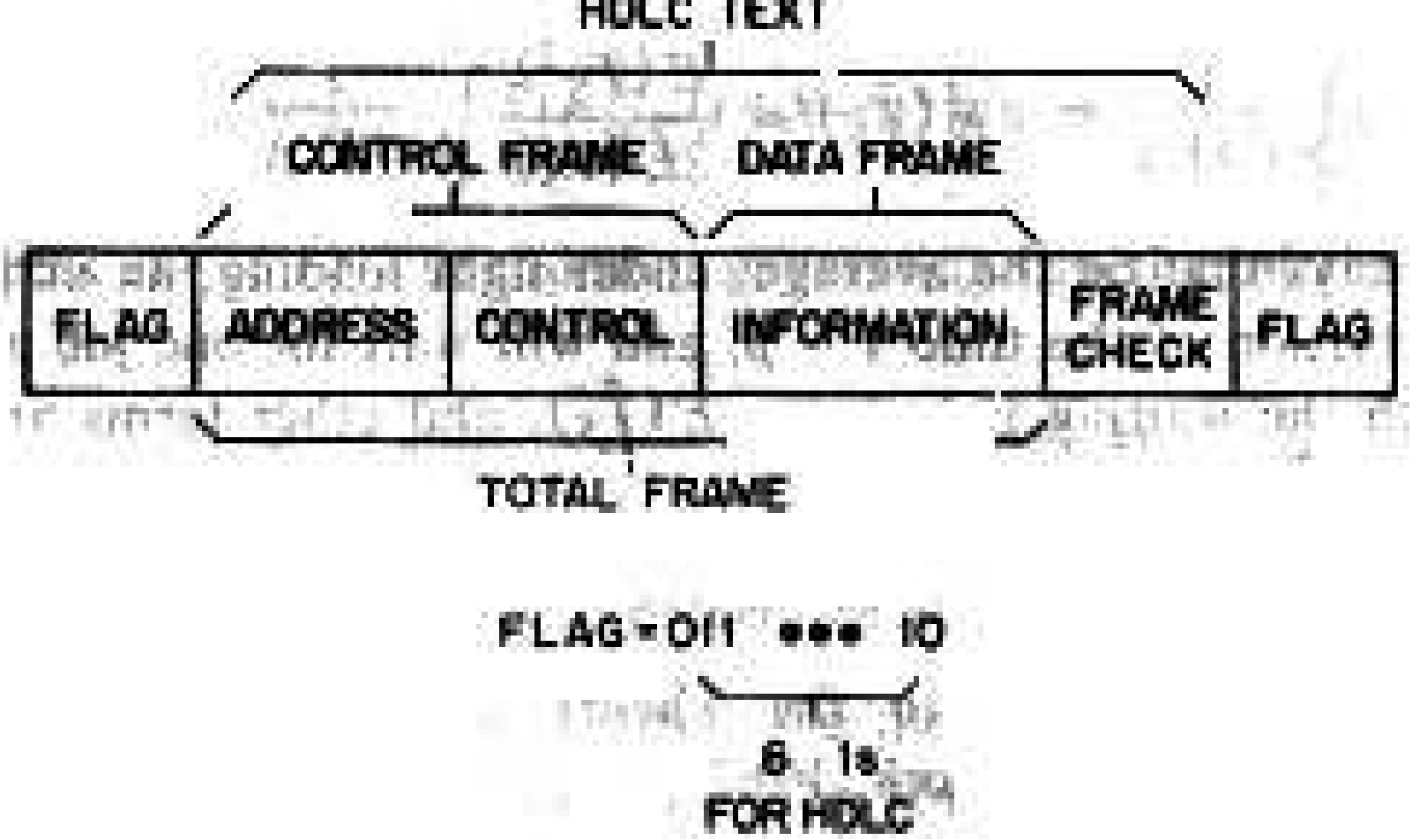

Figure 9.15.SDLC Frame Format

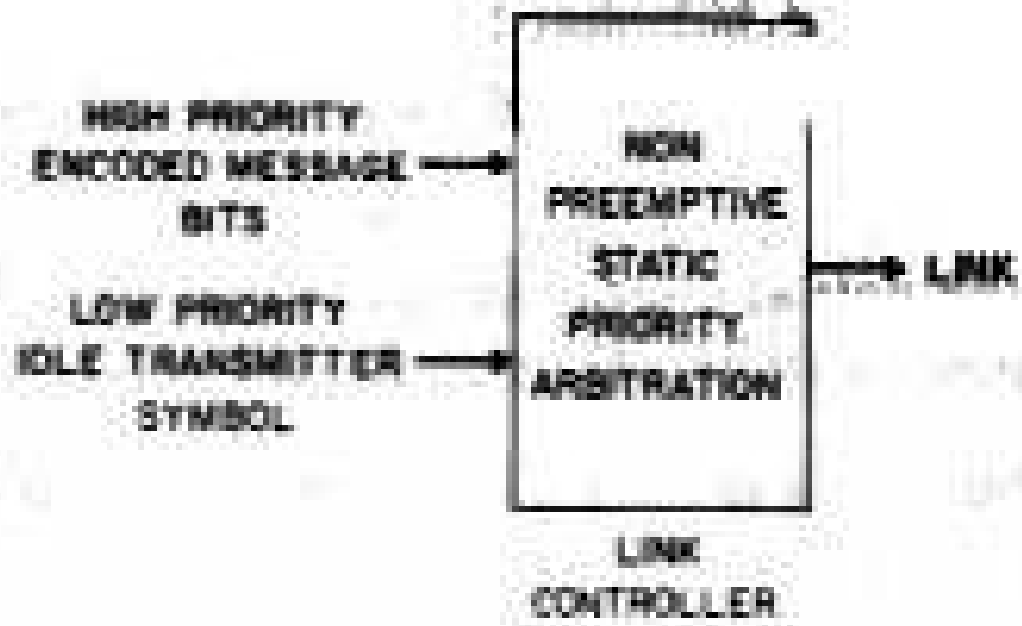

Figure 9.16.Priority Arbitration Queueing Network Block Diagram

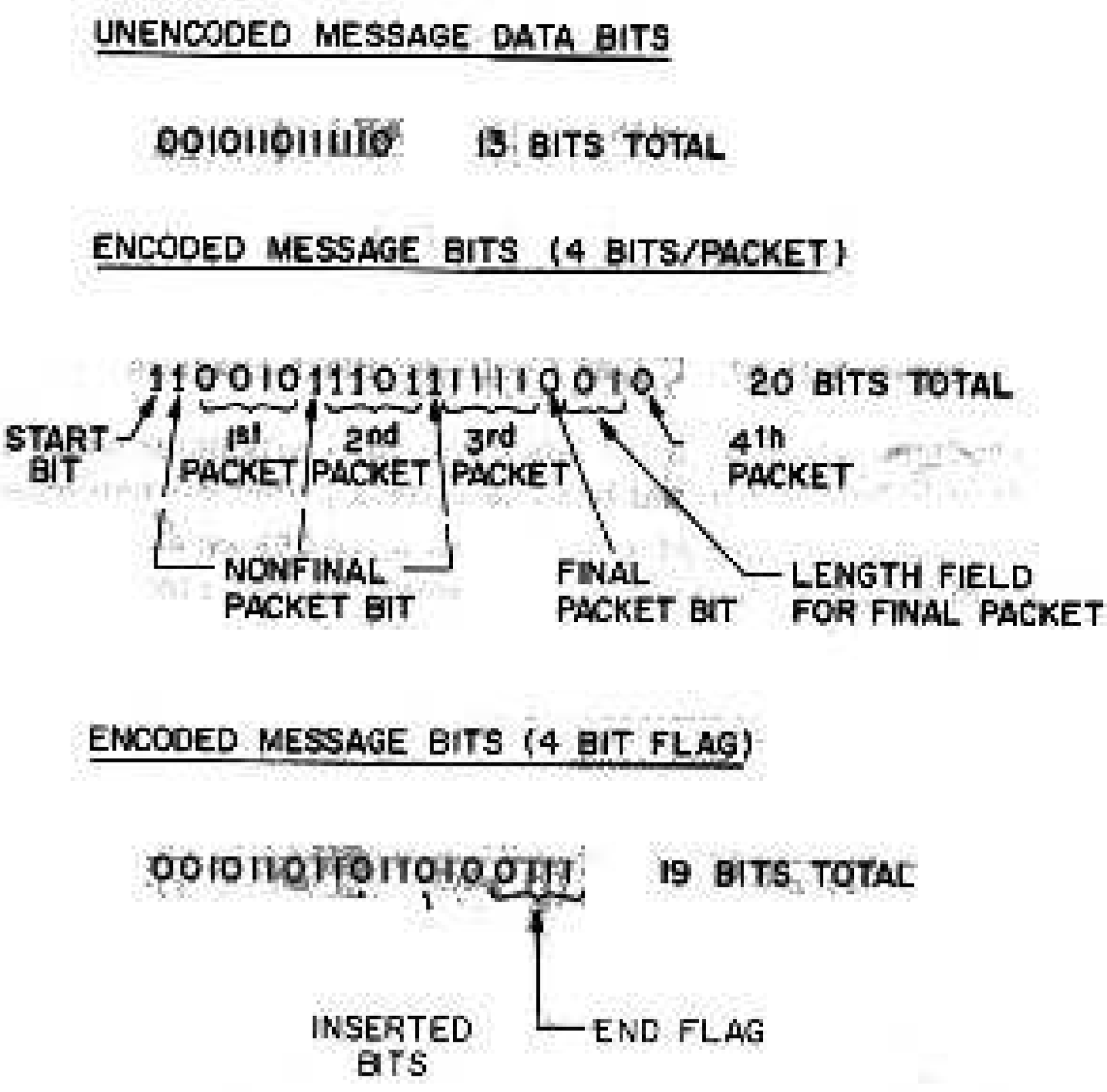

Figure 9.17.Illustrative Bit Stuffing Example: D=13, R=2

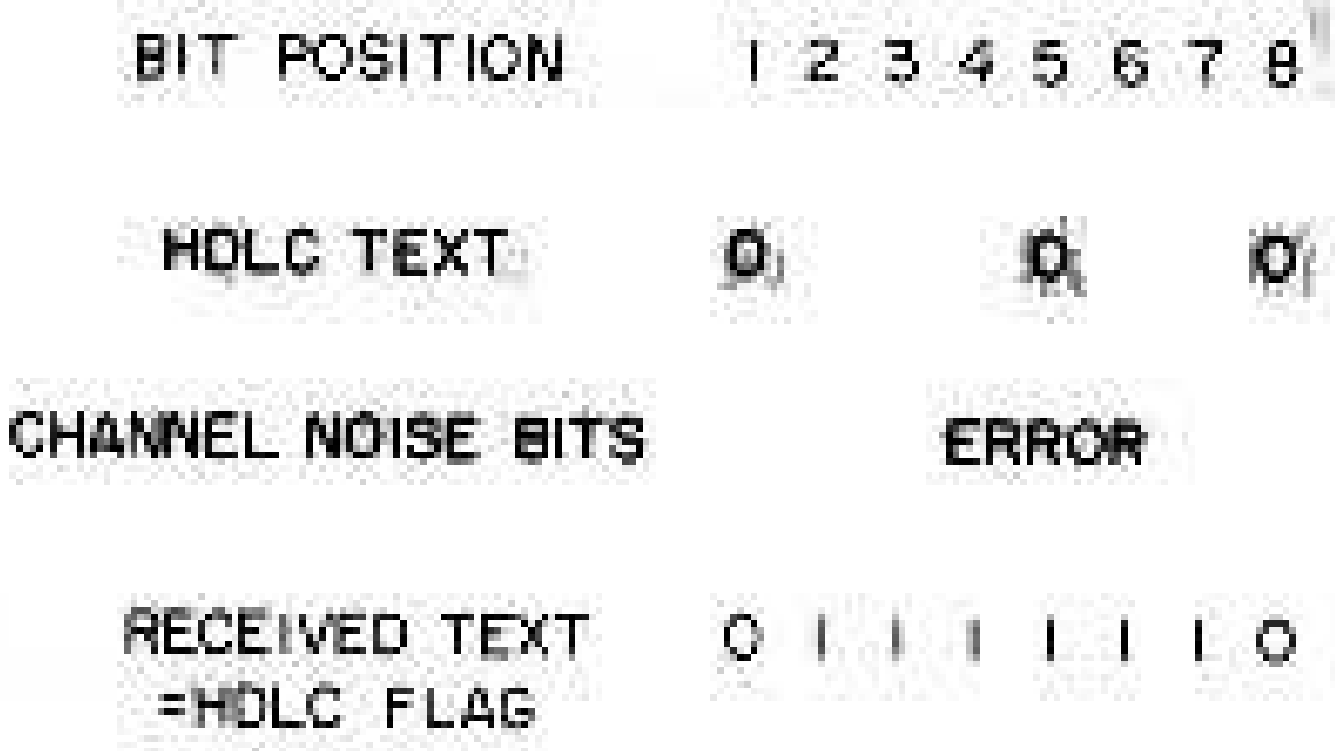

Figure 9.18.Illustrative Spurious Flag from Single Bit Error

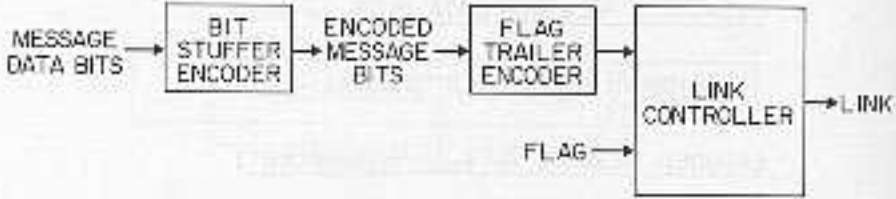

Figure 9.19.SDLC Link Controller Block Diagram