

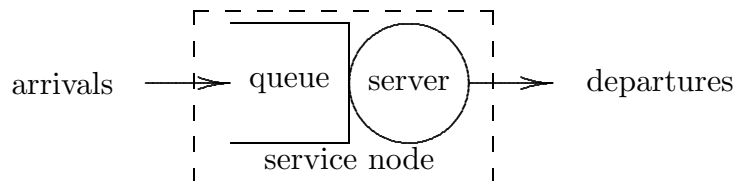
1.2 A Single-Server Queue

In this section we will construct a *trace-driven* discrete-event simulation model (i.e., a model driven by external data) of a single-server service node. We begin the construction at the conceptual level.

1.2.1 CONCEPTUAL MODEL

Definition 1.2.1 A *single-server service node* consists of a *server* plus its *queue*.*

Figure 1.2.1.
Single-server
service node
system diagram.



Jobs (customers) arrive at the service node at random points in time seeking service. When service is provided, the service time involved is also random. At the completion of service, jobs depart. The service node operates as follows: as each (new) job arrives, if the server is busy then the job enters the queue, else the job immediately enters service; as each (old) job departs, if the queue is empty then the server becomes idle, else a job is selected from the queue to immediately enter service. At any time, the state of the server will either be *busy* or *idle* and the state of the queue will be either *empty* or *not empty*. If the server is idle, the queue must be empty; if the queue is not empty then the server must be busy.

Example 1.2.1 If there is just one service technician, the machine shop model presented in Examples 1.1.1 and 1.1.2 is a single-server service node model. That is, the “jobs” are the machines to be repaired and the “server” is the service technician. (In this case, whether the jobs move to the server or the server moves to the jobs is not an important distinction because the repair time is the primary source of delay.)

Definition 1.2.2 Control of the queue is determined by the *queue discipline* — the algorithm used when a job is selected from the queue to enter service. The standard algorithms are:

- FIFO — first in, first out (the traditional computer science queue data structure);
- LIFO — last in, first out (the traditional computer science stack data structure);
- SIRO — service in random order;
- Priority — typically, shortest job first (SJF) or equivalently, in job-shop terminology, shortest processing time (SPT).

The maximum possible number of jobs in the service node is the *capacity*. The capacity can be either *finite* or *infinite*. If the capacity is finite then jobs that arrive and find the service node full will be *rejected* (unable to enter the service node).

* The term “service node” is used in anticipation of extending this model, in later chapters, to a *network* of service nodes.

Certainly the most common queue discipline is FIFO (also known as FCFS — first come, first served). If the queue discipline is FIFO, then the order of arrival to the service node and the order of departure from the service node are the same; there is no passing. In particular, upon arrival a job will enter the queue if and only if the *previous* job has not yet departed the service node. This is an important observation that can be used to simplify the simulation of a FIFO single-server service node. If the queue discipline is not FIFO then, for at least some jobs, the order of departure will differ from the order of arrival. In this book, the *default* assumptions are that the queue discipline is FIFO and the service node capacity is infinite, unless otherwise specified. Discrete-event simulation allows these assumptions to be easily altered for more realistic modeling.

There are two important additional default assumptions implicit in Definition 1.2.1. First, service is *non-preemptive* — once initiated, service on a job will be continued until completion. That is, a job in service cannot be preempted by another job arriving later. Preemption is commonly used with priority queue disciplines to prevent a job with a large service time requirement from producing excessive delays for small jobs arriving soon after service on the large job has begun. Second, service is *conservative* — the server will never remain idle if there is one or more jobs in the service node. If the queue discipline is not FIFO *and* if the next arrival time is known in advance then, even though one or more jobs are in the service node, it may be desirable for a non-conservative server to remain idle until the next job arrives. This is particularly true in non-preemptive job scheduling applications if a job in the service node has a much larger service requirement than the next job scheduled to arrive.

1.2.2 SPECIFICATION MODEL

The following variables, illustrated in Figure 1.2.2, provide the basis for moving from a conceptual model to a specification model. At their arrival to the service node, jobs are indexed by $i = 1, 2, 3, \dots$. For each job there are six associated time variables.

- The *arrival time* of job i is a_i .
- The *delay* of job i in the queue is $d_i \geq 0$.
- The time that job i begins service is $b_i = a_i + d_i$.
- The *service time* of job i is $s_i > 0$.
- The *wait* of job i in the service node (queue and service) is $w_i = d_i + s_i$.
- The time that job i completes service (the *departure time*) is $c_i = a_i + w_i$.

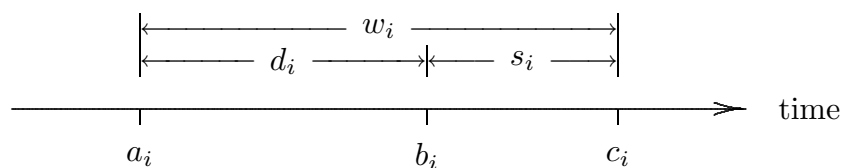


Figure 1.2.2.
Six variables
associated
with job i .

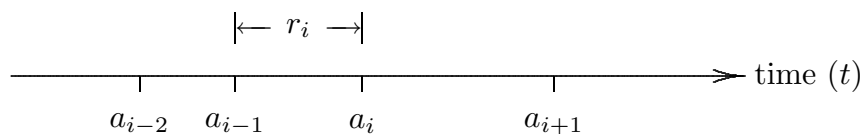
The term “wait” can be confusing; w_i represents the total time job i spends in the service node, *not* just the time spent in the queue. The time spent in the queue (if any) is the delay d_i . In many computer science applications the term *response time* is used. To some authors this means *wait*, to others it means *delay*. Because of this ambiguity, we will generally avoid using the term “response time” choosing instead to consistently use the terminology specified previously. Similarly, we avoid the use of the common terms *sojourn time*, *flow time*, or *system time*, in place of wait.

Arrivals

As a convention, if the service node capacity is finite then rejected jobs (if any) are not indexed. That is, although rejected jobs may be counted for statistical purposes (for example, to estimate the probability of rejection), the index $i = 1, 2, 3, \dots$ is restricted to only those jobs that actually enter the service node.

Rather than specify the *arrival* times a_1, a_2, \dots explicitly, in some discrete-event simulation applications it is preferable to specify the *interarrival* times r_1, r_2, \dots , thereby defining the arrival times implicitly, as shown in Figure 1.2.3 and defined in Definition 1.2.3.

Figure 1.2.3.
Relationship
between
arrival and
interarrival
times.



Definition 1.2.3 The *interarrival time* between jobs $i - 1$ and i is $r_i = a_i - a_{i-1}$. That is, $a_i = a_{i-1} + r_i$ and so (by induction), with $a_0 = 0$ the arrival times are*

$$a_i = r_1 + r_2 + \dots + r_i \quad i = 1, 2, 3, \dots$$

(We assume that $r_i > 0$ for all i , thereby eliminating the possibility of *bulk* arrivals. That is, jobs are assumed to arrive one at a time.)

Algorithmic Question

The following algorithmic question is fundamental. Given a knowledge of the arrival times a_1, a_2, \dots (or, equivalently, the interarrival times r_1, r_2, \dots), the associated service times s_1, s_2, \dots , and the queue discipline, how can the delay times d_1, d_2, \dots be computed?

As discussed in later chapters, for some queue disciplines this question is more difficult to answer than for others. If the queue discipline is FIFO, however, then the answer is particularly simple. That is, as demonstrated next, if the queue discipline is FIFO then there is a simple algorithm for computing d_i (as well as b_i , w_i , and c_i) for all i .

* All arrival times are referenced to the virtual arrival time a_0 . Unless explicitly stated otherwise, in this chapter and elsewhere we assume that elapsed time is measured in such a way that $a_0 = 0$.

Two Cases

If the queue discipline is FIFO then the delay d_i of job $i = 1, 2, 3, \dots$ is determined by when the job's arrival time a_i occurs relative to the departure time c_{i-1} of the previous job. There are two cases to consider.

- **Case I.** If $a_i < c_{i-1}$, i.e., if job i arrives before job $i - 1$ departs then, as illustrated, job i will experience a delay of $d_i = c_{i-1} - a_i$. Job $i - 1$'s history is displayed above the time axis and job i 's history is displayed below the time axis in Figures 1.2.4 and 1.2.5.

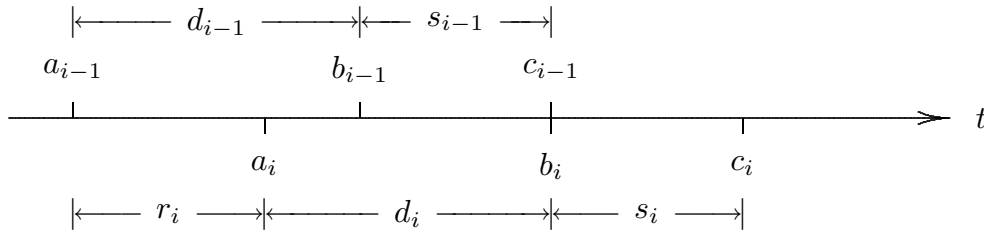


Figure 1.2.4.
Job i arrives before job $i - 1$ departs.

- **Case II.** If instead $a_i \geq c_{i-1}$, i.e., if job i arrives after (or just as) job $i - 1$ departs then, as illustrated, job i will experience no delay so that $d_i = 0$.

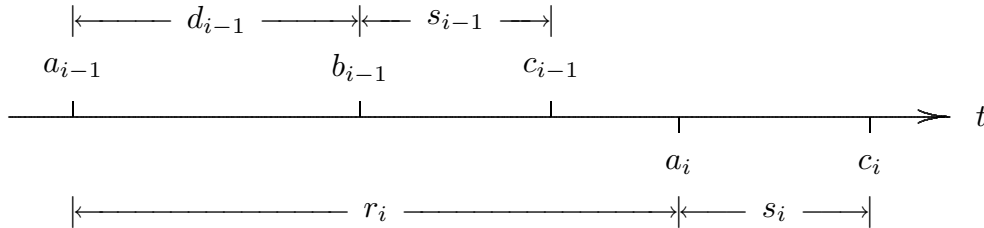


Figure 1.2.5.
Job i arrives after job $i - 1$ departs.

Algorithm

The key point in algorithm development is that if the queue discipline is FIFO then the truth of the expression $a_i < c_{i-1}$ determines whether or not job i will experience a delay. Based on this logic, the computation of the delays is summarized by Algorithm 1.2.1. This algorithm, like all those presented in this book, is written in a C-like pseudo-code that is easily translated into other general-purpose programming languages.

Although it is not an explicit part of Algorithm 1.2.1, an equation can be written for the delay that depends on the interarrival and service times only. That is

$$\begin{aligned} c_{i-1} - a_i &= (a_{i-1} + d_{i-1} + s_{i-1}) - a_i \\ &= d_{i-1} + s_{i-1} - (a_i - a_{i-1}) \\ &= d_{i-1} + s_{i-1} - r_i. \end{aligned}$$

If $d_0 = s_0 = 0$ then d_1, d_2, d_3, \dots are defined by the nonlinear equation

$$d_i = \max\{0, d_{i-1} + s_{i-1} - r_i\} \quad i = 1, 2, 3, \dots$$

This equation is commonly used in theoretical studies to analyze the stochastic behavior of a FIFO service node.

Algorithm 1.2.1 If the arrival times a_1, a_2, \dots and service times s_1, s_2, \dots are known and if the server is initially idle, then this algorithm computes the delays d_1, d_2, \dots in a single-server FIFO service node with infinite capacity.

```

c0 = 0.0;                               /* assumes that a0 = 0.0 */
i = 0;
while ( more jobs to process ) {
    i++;
    ai = GetArrival();
    if ( ai < ci-1 )
        di = ci-1 - ai;                /* calculate delay for job i */
    else
        di = 0.0;                        /* job i has no delay */
    si = GetService();
    ci = ai + di + si;                /* calculate departure time for job i */
}
n = i;
return d1, d2, ..., dn;

```

The `GetArrival` and `GetService` procedures read the next arrival and service time from a file. (An algorithm that does not rely on the FIFO assumption is presented in Chapter 5.)

Example 1.2.2 If Algorithm 1.2.1 is used to process $n = 10$ jobs according to the input indicated below (for simplicity the a_i 's and s_i 's are integer time units, e.g., seconds, minutes, etc.) then the output is the sequence of delays are calculated as:

	i	:	1	2	3	4	5	6	7	8	9	10
read from file	a_i	:	15	47	71	111	123	152	166	226	310	320
from algorithm	d_i	:	0	11	23	17	35	44	70	41	0	26
read from file	s_i	:	43	36	34	30	38	40	31	29	36	30

For future reference, note that the last job arrived at time $a_n = 320$ and departed at time $c_n = a_n + d_n + s_n = 320 + 26 + 30 = 376$.

As discussed in more detail later in this section, it is a straight-forward programming exercise to produce a computational model of a single-server FIFO service node with infinite capacity using Algorithm 1.2.1. The ANSI C program `ssq1` is an example. Three features of this program are noteworthy. (i) Because of its reliance on previously recorded arrival and service time data read from an external file, `ssq1` is a so-called *trace-driven* discrete-event simulation program. (ii) Because the queue discipline is FIFO, program `ssq1` does not need to use a queue data structure. (iii) Rather than produce a sequence of delays as output, program `ssq1` computes four averages instead: the average interarrival time, service time, delay, and wait. These four job-averaged statistics and three corresponding time-averaged statistics are discussed next.

1.2.3 OUTPUT STATISTICS

One basic issue that must be resolved when constructing a discrete-event simulation model is the question of what statistics should be generated. The purpose of simulation is insight and we gain insight about the performance of a system by looking at meaningful statistics. Of course, a decision about what statistics are most meaningful is dependent upon your perspective. For example, from a job's (customer's) perspective the most important statistic might be the average delay or the 95th percentile of the delay — in either case, the smaller the better. On the other hand, particularly if the server is an expensive resource whose justification is based on an anticipated heavy workload, from management's perspective the server's utilization (the proportion of busy time, see Definition 1.2.7) is most important — the larger the better.

Job-Averaged Statistics

Definition 1.2.4 For the first n jobs, the *average interarrival time* and the *average service time* are, respectively*

$$\bar{r} = \frac{1}{n} \sum_{i=1}^n r_i = \frac{a_n}{n} \quad \text{and} \quad \bar{s} = \frac{1}{n} \sum_{i=1}^n s_i.$$

The reciprocal of the average interarrival time, $1/\bar{r}$, is the *arrival rate*; the reciprocal of the average service time, $1/\bar{s}$, is the *service rate*.

Example 1.2.3 For the $n = 10$ jobs in Example 1.2.2, $\bar{r} = a_n/n = 320/10 = 32.0$ and $\bar{s} = 34.7$. If time in this example is measured in seconds, then the average interarrival time is 32.0 seconds per job and the average service time is 34.7 seconds per job. The corresponding arrival rate is $1/\bar{r} = 1/32.0 \cong 0.031$ jobs per second; the service rate is $1/\bar{s} = 1/34.7 \cong 0.029$ jobs per second. In this particular example, the server is not quite able to process jobs at the rate they arrive on average.

Definition 1.2.5 For the first n jobs, the *average delay* in the queue and the *average wait* in the service node are, respectively

$$\bar{d} = \frac{1}{n} \sum_{i=1}^n d_i \quad \text{and} \quad \bar{w} = \frac{1}{n} \sum_{i=1}^n w_i.$$

Recall that $w_i = d_i + s_i$ for all i . Therefore, the average time spent in the service node will be the sum of the average times spent in the queue and in service. That is

$$\bar{w} = \frac{1}{n} \sum_{i=1}^n w_i = \frac{1}{n} \sum_{i=1}^n (d_i + s_i) = \frac{1}{n} \sum_{i=1}^n d_i + \frac{1}{n} \sum_{i=1}^n s_i = \bar{d} + \bar{s}.$$

The point here is that it is sufficient to compute any two of the statistics \bar{w} , \bar{d} , \bar{s} . The third statistic can then be computed from the other two, if appropriate.

* The equation $\bar{r} = a_n/n$ follows from Definition 1.2.3 and the assumption that $a_0 = 0$.

Example 1.2.4 For the data in Example 1.2.2, $\bar{d} = 26.7$ and $\bar{s} = 34.7$. Therefore, the average wait in the node is $\bar{w} = 26.7 + 34.7 = 61.4$. (See also Example 1.2.6.)

In subsequent chapters we will construct increasingly more complex discrete-event simulation models. Because it is never easy to verify and validate a complex model, it is desirable to be able to apply as many *consistency checks* to the output data as possible. For example, although program `ssq1` is certainly not a complex discrete-event simulation model, it is desirable in this program to accumulate \bar{w} , \bar{d} , and \bar{s} *independently*. Then, from the program output the equation $\bar{w} = \bar{d} + \bar{s}$ can be used as a consistency check.

Time-Averaged Statistics

The three statistics \bar{w} , \bar{d} and \bar{s} are *job-averaged* statistics — the data is averaged over all jobs. Job averages are easy to understand; they are just traditional arithmetic averages. We now turn to another type of statistic that is equally meaningful, *time-averaged*. Time-averaged statistics may be less familiar, however, because they are defined by an area under a curve, i.e., by integration instead of summation.

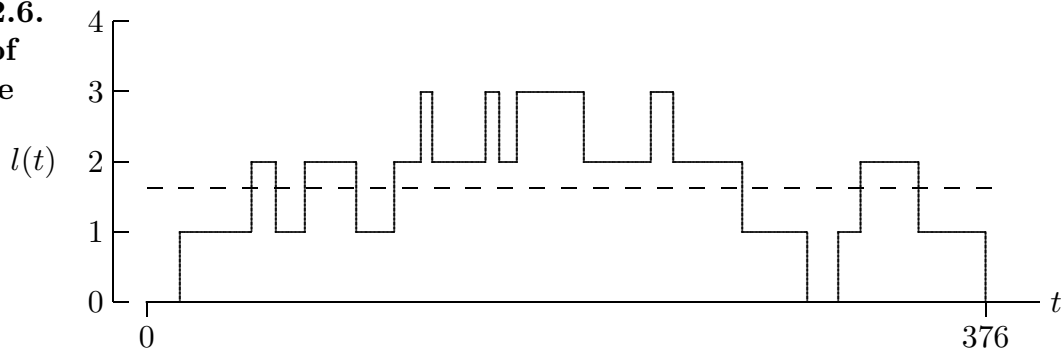
Time-averaged statistics for a single-server service node are defined in terms of three additional variables. At any time $t > 0$:

- $l(t) = 0, 1, 2, \dots$ is the number of jobs in the service node at time t ;
- $q(t) = 0, 1, 2, \dots$ is the number of jobs in the queue at time t ;
- $x(t) = 0, 1$ is the number of jobs in service at time t .

By definition, $l(t) = q(t) + x(t)$ for any $t > 0$.

The three functions $l(\cdot)$, $q(\cdot)$, and $x(\cdot)$ are *piecewise constant*. That is, for example, a display of $l(t)$ versus t will consist of a sequence of constant segments with unit height step discontinuities as illustrated in Figure 1.2.6. (This figure corresponds to the data in Example 1.2.2. The dashed line represents the time-averaged number in the node — see Example 1.2.6.)

Figure 1.2.6.
Number of
jobs in the
service
node.



The step discontinuities are positive at the arrival times and negative at the departure times. The corresponding figures for $q(\cdot)$ and $x(\cdot)$ can be deduced from the fact that $q(t) = 0$ and $x(t) = 0$ if and only if $l(t) = 0$, otherwise $q(t) = l(t) - 1$ and $x(t) = 1$.

Definition 1.2.6 Over the time interval $(0, \tau)$ the *time-averaged number in the node* is

$$\bar{l} = \frac{1}{\tau} \int_0^{\tau} l(t) dt.$$

Similarly, the *time-averaged number in the queue* and the *time-averaged number in service* are

$$\bar{q} = \frac{1}{\tau} \int_0^{\tau} q(t) dt \quad \text{and} \quad \bar{x} = \frac{1}{\tau} \int_0^{\tau} x(t) dt.$$

Because $l(t) = q(t) + x(t)$ for all $t > 0$ it follows that

$$\bar{l} = \bar{q} + \bar{x}.$$

Example 1.2.5 For the data in Example 1.2.2 (with $\tau = c_{10} = 376$) the three time-averaged statistics are $\bar{l} = 1.633$, $\bar{q} = 0.710$, and $\bar{x} = 0.923$. These values can be determined by calculating the areas associated with the integrals given in Definition 1.2.6 or by exploiting a mathematical relationship between the job-averaged statistics \bar{w} , \bar{d} , and \bar{s} and the time-averaged statistics \bar{l} , \bar{q} , and \bar{x} , as illustrated subsequently in Example 1.2.6.

The equation $\bar{l} = \bar{q} + \bar{x}$ is the time-averaged analog of the job-averaged equation $\bar{w} = \bar{d} + \bar{s}$. As we will see in later chapters, time-averaged statistics have the following important characterizations.

- If we were to observe (sample) the number in the service node, for example, at many different times chosen *at random* between 0 and τ and then calculate the arithmetic average of all these observations, the result should be close to \bar{l} .
- Similarly, the arithmetic average of many random observations of the number in the queue should be close to \bar{q} and the arithmetic average of many random observations of the number in service (0 or 1) should be close to \bar{x} .
- \bar{x} must lie in the closed interval $[0, 1]$.

Definition 1.2.7 The time-averaged number in service \bar{x} is also known as the server *utilization*. The reason for this terminology is that \bar{x} represents the proportion of time that the server is busy.

Equivalently, if one particular time is picked *at random* between 0 and τ then \bar{x} is the probability that the server is busy at that time. If \bar{x} is close to 1.0 then the server is busy most of the time and correspondingly large values of \bar{l} and \bar{q} will be produced. On the other hand, if the utilization is close to 0.0 then the server is idle most of the time and the values of \bar{l} and \bar{q} will be small. The case study, presented later, is an illustration.

Little's Equations

One important issue remains — how are job averages and time averages related? Specifically, how are \bar{w} , \bar{d} , and \bar{s} related to \bar{l} , \bar{q} , and \bar{x} ?

In the particular case of an infinite capacity FIFO service node that begins and ends in an idle state, the following theorem provides an answer to this question. See Exercise 1.2.7 for a generalization of this theorem to any queue discipline.

Theorem 1.2.1 (Little, 1961) If the queue discipline is FIFO, the service node capacity is infinite, and the server is idle both initially (at $t = 0$) and immediately after the departure of the n^{th} job (at $t = c_n$) then

$$\int_0^{c_n} l(t) dt = \sum_{i=1}^n w_i \quad \text{and} \quad \int_0^{c_n} q(t) dt = \sum_{i=1}^n d_i \quad \text{and} \quad \int_0^{c_n} x(t) dt = \sum_{i=1}^n s_i.$$

Proof For each job $i = 1, 2, \dots$, define an *indicator function* $\psi_i(t)$ that is 1 when the i^{th} job is in the service node and is 0 otherwise

$$\psi_i(t) = \begin{cases} 1 & a_i < t < c_i \\ 0 & \text{otherwise.} \end{cases}$$

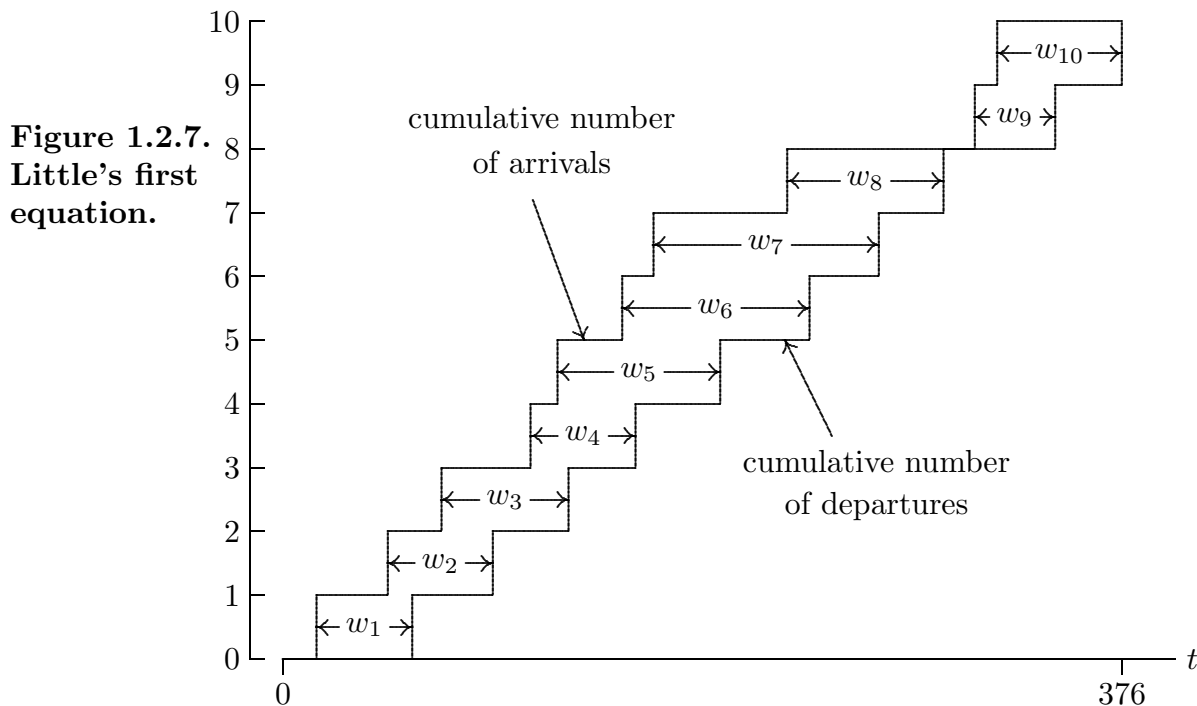
Then

$$l(t) = \sum_{i=1}^n \psi_i(t) \quad 0 < t < c_n$$

and so

$$\int_0^{c_n} l(t) dt = \int_0^{c_n} \sum_{i=1}^n \psi_i(t) dt = \sum_{i=1}^n \int_0^{c_n} \psi_i(t) dt = \sum_{i=1}^n (c_i - a_i) = \sum_{i=1}^n w_i.$$

The other two equations can be derived in a similar way.



Example 1.2.6 Figure 1.2.7 illustrates Little's first equation for the data in Example 1.2.2. The top step function denotes the cumulative number of arrivals to the service node and the bottom step function denotes the cumulative number of departures from the service node. The vertical distance between the two step-functions at any time t is $l(t)$, which was plotted in Figure 1.2.6. The wait times are indicated as the horizontal distances between the risers. In this figure, it is easy to see that

$$\int_0^{376} l(t) dt = \sum_{i=1}^{10} w_i = 614.$$

Little's equations provide a valuable link between the job-averaged statistics \bar{w} , \bar{d} , and \bar{s} and the time-averaged statistics \bar{l} , \bar{q} , and \bar{x} . In Definition 1.2.6 let $\tau = c_n$. Then from Theorem 1.2.1 we have

$$c_n \bar{l} = \int_0^{c_n} l(t) dt = \sum_{i=1}^n w_i = n\bar{w}$$

so that $\bar{l} = (n/c_n)\bar{w}$. Similarly, $c_n \bar{q} = n\bar{d}$ and $c_n \bar{x} = n\bar{s}$. Therefore

$$\bar{l} = \left(\frac{n}{c_n}\right)\bar{w} \quad \text{and} \quad \bar{q} = \left(\frac{n}{c_n}\right)\bar{d} \quad \text{and} \quad \bar{x} = \left(\frac{n}{c_n}\right)\bar{s}$$

which explains how \bar{w} , \bar{d} , and \bar{s} are related to \bar{l} , \bar{q} , and \bar{x} . These important equations relate to *steady-state* statistics and *Little's equations* — for more detail, see Chapter 8.

Example 1.2.7 For the data in Example 1.2.2 the last ($n = 10$) job departs at $c_n = 376$. From Example 1.2.4, $\bar{w} = 61.4$ and therefore $\bar{l} = (10/376)61.4 \cong 1.633$. Similarly, the time-averaged number in the queue and in service are $\bar{q} = (10/376)26.7 \cong 0.710$ and $\bar{x} = (10/376)34.7 \cong 0.923$.

1.2.4 COMPUTATIONAL MODEL

As discussed previously, by using Algorithm 1.2.1 in conjunction with some statistics gathering logic it is a straight-forward programming exercise to produce a computational model of a single-server FIFO service node with infinite capacity. The ANSI C program `ssq1` is an example. Like all of the software presented in this book, this program is designed with readability and extendibility considerations.

Program `ssq1`

Program `ssq1` reads arrival and service time data from the disk file `ssq1.dat`. This is a text file that consists of arrival times a_1, a_2, \dots, a_n and service times s_1, s_2, \dots, s_n for $n = 1000$ jobs in the format

$$\begin{array}{ll} a_1 & s_1 \\ a_2 & s_2 \\ \vdots & \vdots \\ a_n & s_n \end{array}$$

In Chapter 3 we will free this trace-driven program from its reliance on external data by using randomly generated arrival and service times instead.

Because the queue discipline is FIFO, program `ssq1` does not need to use a queue data structure. In Chapter 5 we will consider non-FIFO queue disciplines and some corresponding priority queue data structures that can be used at the computational model level.

Program `ssq1` computes the average interarrival time \bar{r} , the average service time \bar{s} , the average delay \bar{d} , and the average wait \bar{w} . In Exercise 1.2.2 you are asked to modify this program so that it will also compute the time-averaged statistics \bar{l} , \bar{q} , and \bar{x} .

Example 1.2.8 For the datafile `ssq1.dat` the observed arrival rate $1/\bar{r} \cong 0.10$ is significantly less than the observed service rate $1/\bar{s} \cong 0.14$. If you modify `ssq1` to compute \bar{l} , \bar{q} , and \bar{x} you will find that $1 - \bar{x} \cong 0.28$, and so the server is idle 28% of the time. Despite this significant idle time, enough jobs are delayed so that the average number in the queue is nearly 2.0.

Traffic Intensity

The ratio of the arrival rate to the service rate is commonly called the *traffic intensity*. From the equations in Definition 1.2.4 and Theorem 1.2.1 it follows that the observed traffic intensity is the ratio of the observed arrival rate to the observed service rate

$$\frac{1/\bar{r}}{1/\bar{s}} = \frac{\bar{s}}{\bar{r}} = \frac{\bar{s}}{a_n/n} = \left(\frac{c_n}{a_n} \right) \bar{x}.$$

Therefore, provided the ratio c_n/a_n is close to 1.0, the traffic intensity and utilization will be nearly equal. In particular, if the traffic intensity is less than 1.0 and n is large, then it is reasonable to expect that the ratio $c_n/a_n = 1 + w_n/a_n$ will be close to 1.0. We will return to this question in later chapters. For now, we close with an example illustrating how relatively sensitive the service node statistics \bar{l} , \bar{q} , \bar{w} , and \bar{d} are to changes in the utilization and how nonlinear this dependence can be.

Case Study

Sven & Larry's Ice Cream Shoppe is a thriving business that can be modeled as a single-server queue. The owners are considering adding additional flavors and cone options, but are concerned about the resultant increase in service times on queue length. They decide to use a trace-driven simulation to assess the impact of the longer service times associated with the additional flavors and cone options.

The file `ssq1.dat` represents 1000 customer interactions at Sven & Larry's. The file consists of arrival times of groups of customers and the group's corresponding service times. The service times vary significantly because of the number of customers in each group.

The dependence of the average queue length \bar{q} on the utilization \bar{x} is illustrated in Figure 1.2.8. This figure was created by systematically increasing or decreasing each service time in the datafile `ssq1.dat` by a common multiplicative factor, thereby causing both \bar{x} and \bar{q} to change correspondingly. The $(\bar{x}, \bar{q}) \cong (0.72, 1.88)$ point circled corresponds to the data in `ssq1.dat` while the point immediately to its right, for example, corresponds to the same data with each service time multiplied by 1.05. The next point to the right corresponds to each service time multiplied by 1.10. From this figure, we see that even a modest increase in service times will produce a significant increase in the average queue length. A nonlinear relationship between \bar{x} and \bar{q} is particularly pronounced for utilizations near $\bar{x} = 1$. A 15% increase in the service times from their current values will result in a 109% increase in the time-averaged number of customers in queue, whereas a 30% increase in the service times from their current values will result in a 518% increase in the time-averaged number of customers in queue.

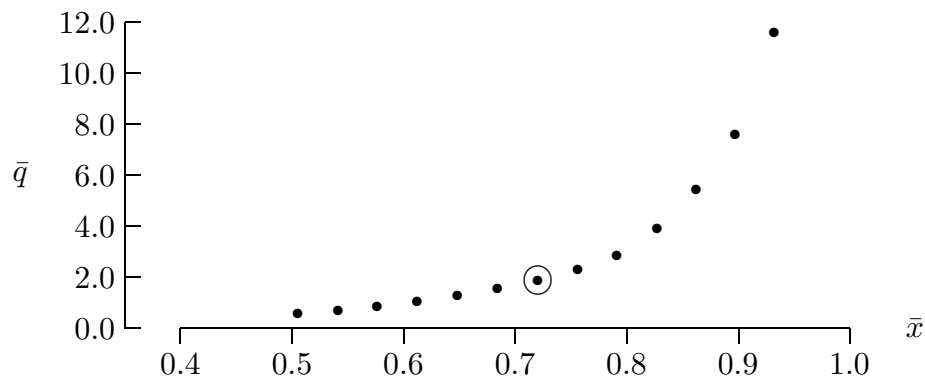


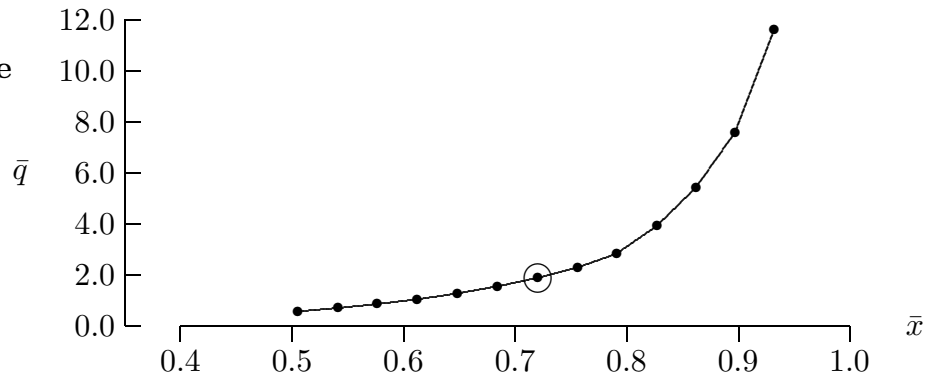
Figure 1.2.8.
Average queue length vs. utilization.

Sven & Larry need to assess the impact of the increased service time associated with new flavors and cones on their operation. If the service times increase by only a modest amount, say 5% or 10% above the current times, the average queue length will grow modestly. The new flavors and cone options may, however, also increase the arrival rate — potentially exacerbating the problem with long lines. If queues grow to the point where the owners believe that customers are taking their ice cream business elsewhere, they should consider hiring a second server. A separate analysis would be necessary to determine the probability that an arriving group of customers will balk (never enter the queue) or renege (depart from the queue after entering) as a function of the queue’s length.

Graphics Considerations

Figure 1.2.8 presents “raw” simulation output data. That is, each \bullet represents a computed (\bar{x}, \bar{q}) point. Because there is nothing inherently discrete about either \bar{x} or \bar{q} , many additional points could have been computed and displayed to produce an (essentially) continuous \bar{q} versus \bar{x} curve. In this case, however, additional computations seem redundant; few would question the validity of the smooth curve produced by connecting the \bullet ’s with lines, as illustrated in Figure 1.2.9. The nonlinear dependence of \bar{q} on \bar{x} is evident, particularly as \bar{x} approaches 1.0 and the corresponding increase in \bar{q} becomes dramatic.

Figure 1.2.9.
Average queue
length vs.
utilization
with linear
interpolation.



Perhaps because we were taught to do this as children, there is a natural tendency to *always* “connect the dots” (interpolate) when presenting a discrete set of experimental data. (The three most common interpolating functions are linear, quadratic, and spline functions.) Before taking such artistic liberties, however, consider the following guidelines.

- If the data has essentially no uncertainty and if the resulting interpolating curve is smooth, then there is little danger in connecting the dots — provided the original dots are left in the figure to remind the reader that some artistic license was used.
- If the data has essentially no uncertainty but the resulting interpolating curve is not smooth then more dots need to be generated to achieve a graphics scale at which smooth interpolation is reasonable.
- If the dots correspond to uncertain (noisy) data then interpolation is not justified; instead, either approximation should be used in place of interpolation, or the temptation to superimpose a continuous curve should be resisted completely.

These guidelines presume that the data is not inherently discrete. If the data is inherently discrete then it is illogical and potentially confusing to superimpose a continuous (interpolating or approximating) curve. Example 1.3.7 in the next section is an illustration of data that is inherently discrete.*

1.2.5 EXERCISES

Exercise 1.2.1^a How would you use the table in Example 1.2.2 to construct the associated $l(t)$ versus t figure? That is, construct an algorithm that will compute *in order* the interlaced arrival and departure times that define the points at which $l(t)$ changes. (Avoid storing a_1, a_2, \dots, a_n and c_1, c_2, \dots, c_n as two arrays, linked lists or external disk files and then merging the two into one due to memory and CPU considerations for large n .)

* Those interested in an excellent discussion and illustration of graphics considerations are encouraged to read the classic *The Visual Display of Quantitative Information* (Tufte, 2001). The author discusses clarity of presentation through uncluttered graphics that maximize information transmission with minimal ink. The accurate display of simulation output will be stressed throughout this text.

Exercise 1.2.2 (a) Modify program `ssq1` to output the additional statistics \bar{l} , \bar{q} , and \bar{x} . (b) Similar to the case study, use this program to compute a table of \bar{l} , \bar{q} , and \bar{x} for traffic intensities of 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, and 1.2. (c) Comment on how \bar{l} , \bar{q} , and \bar{x} depend on the traffic intensity. (d) Relative to the case study, if it is decided that \bar{q} greater than 5.0 is not acceptable, what systematic increase in service times would be acceptable? Use *d.dd* precision.

Exercise 1.2.3 (a) Modify program `ssq1` by adding the capability to compute the maximum delay, the number of jobs in the service node at a specified time (known at compile time) and the proportion of jobs delayed. (b) What was the maximum delay experienced? (c) How many jobs were in the service node at $t = 400$ and how does the computation of this number relate to the proof of Theorem 1.2.1? (d) What proportion of jobs were delayed and how does this proportion relate to the utilization?

Exercise 1.2.4 Complete the proof of Theorem 1.2.1.

Exercise 1.2.5 If the traffic intensity is less than 1.0, use Theorem 1.2.1 to argue why for large n you would expect to find that $\bar{l} \cong \lambda \bar{w}$, $\bar{q} \cong \lambda \bar{d}$, and $\bar{x} \cong \lambda \bar{s}$, where the observed arrival rate is $\lambda = 1/\bar{r}$.

Exercise 1.2.6 The text file `ac.dat` consists of the arrival times a_1, a_2, \dots, a_n and the departure times c_1, c_2, \dots, c_n for $n = 500$ jobs in the format

a_1	c_1
a_2	c_2
\vdots	\vdots
a_n	c_n

(a) If these times are for an initially idle single-server FIFO service node with infinite capacity, calculate the average service time, the server's utilization and the traffic intensity. (b) Be explicit: for $i = 1, 2, \dots, n$ how does s_i relate to a_{i-1} , a_i , c_{i-1} , and c_i ?

Exercise 1.2.7^a State and prove a theorem analogous to Theorem 1.2.1 but valid for *any* queue discipline. *Hint*: in place of c_n use $\tau_n = \max\{c_1, c_2, \dots, c_n\}$. For a conservative server prove that τ_n is *independent* of the queue discipline.

Exercise 1.2.8 (a) Similar to Exercise 1.2.2, modify program `ssq1` to output the additional statistics \bar{l} , \bar{q} , and \bar{x} . (b) By using the arrival times in the file `ssq1.dat` and an appropriate *constant* service time in place of the service times in the file `ssq1.dat`, use the modified program to compute a table of \bar{l} , \bar{q} , and \bar{x} for traffic intensities of 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, and 1.2. (c) Comment on how \bar{l} , \bar{q} , and \bar{x} depend on the traffic intensity.

Exercise 1.2.9^a (a) Work Exercises 1.2.2 and 1.2.8. (b) Compare the two tables produced and explain (or conjecture) why the two tables are different. Be specific.