CHAPTER VI
COMPUTER NETWORKS

New developments in queueing theory, relevant to performance analysis, extend the class of problems, amenable to analytical treatments. The earlier work by Mitra [41] has given a break through in the study of a specific limiting operation that gives open networks to closed networks. Here we give a brief account of a qualitative characterisation of processor — sharing discrimination against long jobs in computer networks in section 6.1. In section 6.2 a newly introduced approach suggested by Mckenna [39] to the problem of computing the partition function of closed stochastic networks from which we could get different characterisation of stationary distributions. Computer systems are multiple resource systems consisting of on line terminals, communication lines, and controllers with the line concentrators. Thus a computer system is a network of queues. A detailed account of M/G/1, processor-sharing system and a special type open Jackson network as studied by Ramaswami and Latouche [51] have been considered in section 6.3. We have developed [67] a method for calculation of mean delay using Fuzzy and neural networks and arrived at a better and easier computational procedure in the last section.

SECTION 6.1. MODELS OF SHARED-PROCESSOR SYSTEMS
We compute and analyze the waiting time distribution of a well-known [34] physical model, consisting of a bank of terminals in series with a CPU which feeds back to the terminals. Each terminal spends its time alternating between the ‘think’ mode and the ‘waiting’ mode; in the former mode it independently generates jobs, while in the waiting mode the job, now transferred to the CPU, contends with other
jobs for service. On completion of service the job returns to the terminal and a new cycle resumes.

The model and the problem has three main distinguishing features. First, the processing discipline is ‘processor sharing’ [34, 31, 3]. Second, the network is closed [34, 31, 3]. Third, the analytical problem concerns not the stationary distribution of jobs in the network, but the waiting time and its offshoots.

In the processor-sharing discipline there is no overt queueing because all, say $n$, jobs present in the CPU simultaneously receive service at $1/n$ times the rate given to a single job by the processor. This discipline is the limiting case of the round robin discipline, as the time quantum given to each job becomes arbitrarily small. Naturally, the rate given to a single job fluctuates with time and, importantly, the waiting time of a particular job depends not only on the jobs already in the CPU at its time of arrival there but also on subsequent arrivals. This is in contrast to the situation with the first-come-first-served discipline for which, therefore, the waiting time problem is considerably simpler. Processor-sharing favours short jobs and discriminates against long jobs.

![Fig. 6.1.1 Physical Model](image-url)
The network is closed, so that the total number of current or active jobs is constant at \( N \), the number of terminals. We refer to \( N \) as the population. The constancy arises because at any time each terminal has one associated current job which may be either at the think node or the CPU node. The network closure introduces well-known complexities.

While the waiting time distribution is a prime objective, a number of related distributions and their moments are also quite important to bound and estimate the waiting time statistics.

In the model the required service time is independent, identically and exponentially distributed for all jobs. The unit of time is selected to give unity as the mean service time. The ‘think’ times are also independent, identically and exponentially distributed with mean \( 1/p \). Thus,

\[
p = \text{Mean required service time/Mean think time.}
\]

We also define

\[
\rho \triangleq Np
\]

which we expect to reflect in some sense the traffic intensity

Whenever we mention distributions of number of jobs in the network, the stationary distribution being the most prominent, it is assumed that the population is \((N-1)\). This is one less that the population assumed for all the waiting time problems.

The birth-and-death equations given below do not depend on whether the discipline is processor-sharing or first-come-first-served. For \( i = 1, 2, \ldots, N \),
let \( \pi_i(t) \) = Pr[(i-1) jobs in CPU at time t for population = N-1].

Then
\[
\frac{d}{dt} \pi'(t) = \pi'(t) F
\]

where \( F \) is a tridiagonal matrix: for 1 \( \leq i \leq N, \)
\[
F_{i,i-1} = 1, \quad F_{i,i+1} = (N-i)p, \quad F_{ii} = -(F_{i,i-1} + F_{i,i+1}).
\]

Regardless of the initial distribution in (1), as \( t \to \infty \)
\[
\pi(t) \to \pi: \pi' F = 0'
\]

Thus, for \( i = 1, 2, ..., N. \)

\[
\pi_i = \{ \text{stationary probability of (i-1) jobs in the CPU for population = N-1}, \}
\]
\[
= \begin{cases} 
\frac{N-1}{i-1} (i-1)! p^{i-1} / \left( \sum_{j=1}^{N-i} \frac{N-1}{j-1} (j-1)! p^{j-1} \right) 
\end{cases}
\]

We employ the notation \( \bar{n}_s \) and \( \sigma^2_s \) for the equilibrium mean and variance, respectively, of the number of jobs in the CPU. Thus \( \bar{n}_s = \pi'b - 1. \)

\[
\sigma^2_s = (N-1) - \frac{1 - (N-2)p}{p} \bar{n}_s - (\bar{n}_s)^2.
\]

The basic equations for the waiting time

Let the random variable \( \widetilde{W} \) denote the time spent by a particular (also referred to as 'tagged') job in the processing node for a closed system of population N in equilibrium. The details are obtained by conditioning the distribution of the waiting time random variable on at most two variables, namely,
\( \tilde{Q} \): number of jobs at CPU just after arrival there of tagged job.
\( \tilde{T} \): required service time of tagged job.

The process of removing the conditioning is aided by the independence of \( \tilde{Q} \) and \( \tilde{T} \).

**Waiting time conditioned on \( \tilde{Q} \) and \( \tilde{T} \)**

Let

\[
G_n(t, u) = \Pr[\tilde{W} > u | \tilde{Q} = n, \tilde{T} = t], \quad t > 0, u > 0.
\]

As the tagged job is included in the count for \( \tilde{Q} \) and the population is \( N \), we have that \( n = 1, 2, \ldots, N \). On account of processor-sharing we have for \( n = 2, 3, \ldots, (N-1) \) as \( \Delta u \to 0 \)

\[
G_n(t, u+\Delta u) = (n-1) \frac{\Delta u}{n} G_{n-1}(t, u) + [1 - \Delta u \{(N-n)p \}
\]

\[
+ \frac{n-1}{n} \sum \{ G_n(t - \frac{\Delta u}{n}, u) \}
\]

\[
+ (N-n)p \Delta u G_{n+1}(t, u) + O(\Delta u^2).
\]

Hence in the limit,

\[
\frac{\partial}{\partial \Delta u} G_n(t, u) + n \frac{\partial}{\partial t} G_n(t, u) = (n-1) G_{n-1}(t, u) - ((N-n)p(n+1)) G_n(t, u)
\]

\[
+ n (N-n) p G_{n+1}(t, u).
\]

After deriving the end equations corresponding to \( n=1 \) and \( n=N \), we may arrange these equations compactly to be as follows in vector notation,

\[
\frac{\partial}{\partial t} G(t, u) + B \frac{\partial}{\partial u} G(t, u) = AG(t, u) \quad \ldots \ldots (3)
\]
The $N \times N$ matrix $A$ is a transition rate matrix and tridiagonal. The terms respectively above and below the diagonal are given special symbols: for $1 \leq i \leq N$

$$A_{i,i+1} = \gamma_{i-1} \triangleq p_i (N-i); \quad A_{i,i-1} = \delta_{i-1} \triangleq i-1;$$

$$A_{i,i} = -(A_{i,i-1} + A_{i,i+1}).$$

The matrix $B$ is diagonal with integer coefficients:

$$B = \text{diag } \{b\} = \text{diag } \{1, 2, \ldots, N\}.$$  

We may verify that the boundary conditions to (3) are as given below

$$G(t,u) = \hat{1} \text{ if } u < t.$$  

$$G(t,t^+) = \hat{1} - e^{(N-1)t} e_1.$$  

$$G(t,u) = 0 \text{ if } u > Nt.$$  

Here $e_1$ is the vector with 1 as its leading component and 0 elsewhere; also, all components of $\hat{1}$ are 1.

**Waiting time conditioned on $\tilde{Q}$**

Let

$$P_n(u) \triangleq P_r[\tilde{W} > u | \tilde{Q} = n], \quad u \geq 0; \quad n = 1, 2, \ldots, N.$$  

Removing the conditioning on $T$ in $G(t, u)$ should give $P(u)$, the vector with components $P_n(u)$:

$$P(u) = \int_0^{\infty} e^{t} G(t, u) \, dt.$$  

Simple manipulations on (3) give, after taking note of the boundary conditions,

$$\frac{d}{du} P(u) = MP(u) : P(0) = \hat{1}$$

where, $M = B^{-1} (A-I)$.
The row sums of the $N \times N$ matrix $M$ are negative. However, the matrix is tridiagonal. Special symbols are used to denote the terms above and below the diagonal: for $1 \leq i \leq N$,

$$M_{i,i-1} = \xi_i, \quad M_{i,i+1} = \tau_i, \quad M_{i,i} = -(1 + \tau_i).$$

$$= (i - 1)/i; \quad = p(N-i);$$

The formal solution to (4) is

$$P(u) = e^{Mu} \hat{I}.$$ 

**Moments conditioned on $\tilde{Q}$ and $\tilde{T}$**

It is shown below that ordinary differential equations govern the moments of the waiting time conditioned on $\tilde{Q}$ and $\tilde{T}$.

For $t \geq 0$, $n = 1, 2, \ldots, N$ and $j = 1, 2, \ldots$ let

$$m_{n}^{(j)}(t) = \mathbb{E}[W_{n}^{j} | \tilde{Q} = n, \tilde{T} = t]$$

and

$$m^{(j)}(t) = [m_{1}^{(j)}(t), m_{2}^{(j)}(t), \ldots, m_{N}^{(j)}(t)].$$

In words, $m_{n}^{(j)}(t)$ is the $j^{th}$ moment of the waiting time conditioned on the number of jobs in the CPU just after arrival there of the tagged job being $n$ and on the required service time for the tagged job being $t$.

As a general convention, the subscript refers to the $\tilde{Q}$ conditioning and the argument to the $\tilde{T}$ conditioning. Absence of one or both indicates the absence of the corresponding conditioning.

From the respective definitions and boundary conditions on $G$,

$$m^{(j)}(t) = - \int_{t^{+}}^{Nt} \frac{\partial}{\partial u} G(t, u) \, du + t^{j} [\hat{I} - G(t, t^{+})]$$
First integrate by parts, and then differentiate with respect to t. With the help of (3) we obtain

\[ m^{(0)}(t) = 1, \quad t \geq 0 \]

\[
\frac{d}{dt} m^{(j)}(t) = A m^{(j)}(t) + j B m^{(j-1)}(t); \quad m^{(j)}(0) = 0
\]

\[ t \geq 0, \quad j = 1, 2, \ldots \]

Thus,

\[ m^{(j)}(t) = j \int_0^t e^{A(t-\tau)} B m^{(j-1)}(\tau) \, d\tau. \]

**Moments conditioned on \( \tilde{Q} \)**

For \( n = 1, 2, \ldots, N \) and \( j = 1, 2, \ldots \), let

\[ m_n^{(j)} = \mathbb{E}[\tilde{W}^j | \tilde{Q} = n] \]

and,

\[ m^{(j)} = [m_1^{(j)}, m_2^{(j)}, \ldots, m_N^{(j)}]' . \]

By definition

\[ m^{(j+1)} = \int_0^\infty \tau^{j+1} \frac{d}{d\tau} P(\tau) \, d\tau. \]

Simple manipulations and invoking (4) will give

\[ m^{(j+1)} = -(j+1) M^{-1} m^{(j)}; \quad m^{(0)} = \hat{1} \]

\[ m^{(j)} = (-1)^j j! M^2 \hat{1}, \quad j = 0, 1, \ldots \]

**A result that allows the conditioning on \( \tilde{Q} \) to be removed**

An important recent result due to Sevcik and Mitrani [56] is invoked to remove the conditioning on \( \tilde{Q} \). Their result specialized to the present context states that in the network of population \( N \) the equilibrium probability of there being \( n \) jobs
in the CPU just after the arrival of the tagged job, is also the stationary probability of \((n - 1)\) jobs in the CPU in a network of population \((N-1)\). That is,

**Result 6.1.1. (Mitra [41])**

\[
\Pr [\tilde{Q} = n] = \pi_n, \quad n = 1, 2, \ldots, N
\]

where \(\pi_n\) is as in (2).

Combining the above result with the results pertaining to waiting time conditioned on \(\tilde{Q}\) we obtain

\[
\Pr [\tilde{W} > u] = \sum_{n=1}^{N} \pi_n \Pr [\tilde{W} > u | \tilde{Q} = n] = \pi' P(u)
\]

and thus, we have \(\Pr [\tilde{W} > u] = \pi' e^{Mu} \tilde{I}\).

**Result 6.1.2. (Mitra [41])**

\[
\pi' A = 0'
\]

By virtue of result 6.1.2, we note that \(m^{(0)} = \pi' m^{(0)}\) and \(\pi'M^j = -\pi'BM^{j+1}\).

Using this we obtain the following unconditional moments.

If \(m^{(0)} = E[\tilde{W}^j]\), \(j = 1, 2, \ldots\)

then \(m^{(0)} = (-1)^{j+1} j! \pi' BM^{j+1} \tilde{I}\). ... (5)

**Moments conditioned on \(\tilde{T}\)**

To obtain the moment conditioned on \(\tilde{T}\), we remove the conditioning on \(\tilde{Q}\) by making use of Result 6.1.1 and simplify it further by using Result 6.1.2. This procedure yields
\[ m^{(j)}(t) = E[W^j | T = t], \quad j = 1, 2, \ldots \]

\[ = j \int_0^t \pi' B m^{(j-1)}(\tau) \, d\tau \quad \cdots (6) \]

where \( m^{(0)}(\cdot) = \mathbb{I}. \)

**First and second moments**

Put \( j = 1 \) in (5), we get the first moment

\[ E[w] = m^{(1)} = (-1)^{j+1} 1! \pi' BM^{j+1} \mathbb{I} = \pi' BI = \pi' b \]

\[ = \pi'b - 1 + 1 = \bar{n}_b + 1 \text{ where } \bar{n}_b = \pi'b - 1 \quad \cdots (7) \]

Also put \( j = 1 \) in (6) we get

\[ E[W/t] = m^{(1)}(t) = 1 \int_0^t \pi' BM^{1-1}(\tau) \, d\tau \]

\[ = \int_0^t \pi' BI \, d\tau = b \pi' t = t \pi'b = t E[W] \quad \cdots (8) \]

Put \( j = 2 \), in (5) we get the second moment

\[ E[W^2] = m^{(2)} = (-1)^{2+1} 2! \pi' BM^{2+1} \mathbb{I} = (-2) \pi' BM^{-1} \mathbb{I} \]

\[ = (-2) \pi' B(A-I)^{-1} BI = 2 \pi' B(I-A)^{-1} b \]

Also put \( j = 2 \) in (6) we get

\[ E[W^2|t] = m^{(2)}(t) = 2 \int_0^t \pi' BM^{(2-1)}(\tau) \, d\tau \]

\[ = 2 \pi' B \int_0^t [M^1(\tau)] \, d\tau \]

\[ = 2 \pi' B \int_0^t \int_0^s e^{A(s-\tau)} BM^{(0)}(\tau) \, dS \, d\tau \]

\[ = 2 \pi' B \int_0^t \int_0^s e^{A(s-\tau)} BI \, ds \, d\tau \]

\[ = 2 \pi' B \int_0^t \int_0^s e^{\lambda(s-\tau)} \, d\tau \, ds \, b \quad \cdots (9) \]
Open networks

The physical model of the open networks is composed of a single CPU operating under the processor-sharing discipline and subject to service demands from the outside. The job arrival process is stipulated to be Poisson with a parameter \( \rho^{(0)} \) (\( \rho^{(0)} < 1 \)) for the arrival rate, the required service time for the jobs is the same as the closed networks, namely, exponentially distributed with unity mean.

Birth-and-death equations

As previously let

\[ \pi_i(t) = \Pr [(i-1) \text{ jobs in CPU at time } t], \quad i = 1, 2, \ldots \]

It is easily shown that in vector notation,

\[
\frac{d}{dt} \pi'(t) = \pi' F^{(0)}
\]

where the infinite dimensional, tridiagonal matrix \( F^{(0)} \) is

\[
F^{(0)}_{i,i-1} = \hat{1}; \quad F^{(0)}_{i,i+1} = \rho^{(0)}; \quad F^{(0)}_{i,i} = - (F^{(0)}_{i,i-1} + F^{(0)}_{i,i+1}).
\]

Also, the stationary probabilities \( \pi_i \pi' F^{(0)} = 0' \) are

\[
\pi_i = (1 - \rho^{(0)}) \{\rho^{(0)}\}^{i-1}, \quad i = 1, 2, \ldots \quad \ldots (10)
\]

Waiting time

Let \( P_n(u), \ n = 1, 2, \ldots \) be defined as in (4) for the closed network. In vector form

\[
\frac{d}{du} P(u) = (B^{(0)})^{-1} \{A^{(0)} - I\} P(u); \quad P(0) = 1
\]
where $B^{(0)} = \text{diag} \{1, 2, 3, \ldots\}$ .... (11)

and $A^{(0)}$ is the tridiagonal rate matrix

$$A^{(0)}_{i,i-1} = i - I, \quad A^{(0)}_{i,i+1} = i, \quad A^{(0)}_{i,i} = - (A^{(0)}_{i,i-1} + A^{(0)}_{i,i+1}), \quad i \geq 1$$

In an open network the removal of the conditioning on $Q$, the jobs in the CPU just after arrival there of the tagged job, is quite straightforward. It has been known for some time that

$$\Pr[Q = n] = \pi_n$$

where the right hand side is as in (10).

Combining the above with (11) gives the formal solution

$$\Pr[\hat{W} > u] = \pi' \exp \{M^{(0)} u\} \hat{I}$$

**Open networks from closed networks by a limiting procedure**

The key point about the two sets of equations derived above is that they may be obtained from the equations of closed networks through the following limiting procedure:

$$N \rightarrow \infty, \quad p \rightarrow 0, \quad N p = \rho^{(0)}.$$ .... (12)

That is, the matrices $F^{(0)}$, $A^{(0)}$, $M^{(0)}$ may be obtained from the matrices $F$, $A$, $M$ by (12). Also, we may verify that

$$\pi_i |_{\text{closed}} \rightarrow \pi_i |_{\text{open}}$$

In words therefore the equations for the closed network approach those of the open network as the number of terminals approaches infinity and the mean think time
approaches infinity in such a manner that the ratio equals the arrival rate in the open system.

This observation justifies a procedure wherein, given a closed network we connect it with a 'naturally associated' open system (with arrival rate \( Np \)). For large enough \( N \) and \( \rho \) small we expect the closed system and its naturally associated open counterpart to be close.

SECTION 6.2. MARKOVIAN QUEUEING NETWORKS

Let \( p \) be the number of classes of jobs and reserve the symbol \( j \) for indexing class. Hence when the index for summation or multiplication is omitted it is understood that the missing index is \( j \) where \( 1 \leq j \leq p \). A total of \( s \) service centers are allowed. We will find it natural to distinguish the centers of Types 1, 2 and 4 which have queueing from the remaining centers of Type 3 which do not. Thus, centers 1 through \( q \) will be the queueing centers while \( (q+1) \) through \( s \) will be the Type 3 centers, which have also been called think nodes and infinite server nodes. We reserve the symbol \( i \) for indexing centers. Also, whenever class and center indices appear together, the first always refers to class.

Let the equilibrium probability of finding \( n_{ji} \) jobs of class \( j \) at center \( i \), \( 1 \leq j \leq p \), \( 1 \leq i \leq s \), be \( \pi(y_1, y_2, \ldots, y_s) \), where

\[
y_i = (n_{1i}, n_{2i}, \ldots, n_{pi}), \quad 1 \leq i \leq s.
\]

Closed networks are characterized by conservation of jobs in each class. That is, the population of jobs of the \( j \)th class is constant at \( K_j \), say. The well-known results on closed networks with the product form in its stationary distribution may be given in the following form:
\[
\pi(y_1, \ldots, y_s) = \frac{1}{G} \prod_{i=1}^{s} \pi_i(y_i) \quad \ldots \quad (14)
\]

where, \( \pi_i(y_i) = (\sum n_{ji})! \prod_{j=1}^{n_{ji}} \left( \frac{\rho_j}{n_{ji}!} \right), \quad 1 \leq i \leq q, \quad \ldots \quad (15) \)

\[
= \pi \left( \frac{\rho_j}{n_{ji}!} \right), \quad (q+1) \leq i \leq s.
\]

In the above formulae we have taken into account the previously stated assumption, namely, for the first-come-first-served discipline in Type 1 centers the service rate is independent of the number of jobs in queue. Also, in (15),

\[
\rho_j = \frac{\text{expected number of visits of class } j \text{ jobs to center } i}{\text{service rate of class } j \text{ jobs in center } i} \quad \ldots \quad (16)
\]

where the numerator is obtained from the given routing matrix by solving for the eigenvector corresponding to the eigenvalue at 1.

In (14) \( G \) is the partition function and it is explicitly

\[
G(K) = \sum_{n_1=k_1}^{\hat{\Gamma}_{n_1}} \ldots \sum_{n_p=k_p}^{\hat{\Gamma}_{n_p}} \prod_{i=1}^{s} \pi_i(y_i) \quad \ldots \quad (17)
\]

where we have written \( \hat{\Gamma}_{n_j} \) for \( \sum_{j=1}^{q} n_{ji} \) and the condition \( \hat{\Gamma}_{n_j} = k_j \) to indicate the conservation of jobs in each class. Thus,

\[
G(K) = \sum \ldots \sum \left[ \prod_{i=1}^{q} (\sum n_{ji})! \prod_{j=1}^{n_{ji}} \left( \frac{\rho_j}{n_{ji}!} \right) \right] \left[ \prod_{i=q+1}^{s} \left( \frac{\rho_j}{n_{ji}!} \right) \right] \quad \ldots \quad (18)
\]
Result 6.2.1

\[ G(K) = \prod_{j \in I} \rho_j^n / \prod_{j} K_j ! \int_{Q^+} e^{\sum_{j} (\delta_{ji} + r_j u_j)^{k_j}} du \quad \ldots \ldots (19) \]

\[ = \prod_{j \in I} \rho_j^n / \prod_{j} K_j ! \int_{Q^+} e^{-Nf(z)} dz \quad \ldots \ldots (20) \]

where \( f(z) = \sum_{j} \beta_j \log (\delta_{ji} + \Gamma_j z) \).

**Proof**

Consider Euler's integral representation for the factorial,

\[ n! = \int_{0}^{\infty} e^{-u} u^n \, du. \quad \ldots \ldots (22) \]

Using the above representation we have

\[ (\sum n_{ji}) ! = \int_{0}^{\infty} e^{-u_i} \prod u_i^{n_{ji}} \, du_i, \quad i = 1, 2, \ldots, q \quad \ldots \ldots (23) \]

Substituting in (18) we get

\[ G = \int_{0}^{\infty} \cdots \int_{0}^{\infty} \exp \left[ -\sum u_i \sum_{i=1}^{q} \sum_{\Gamma_{n} = k_p} \prod \left\{ \prod_{j=1}^{n_{ji}} \left( \frac{\rho_{ji} u_i^{n_{ji}}}{n_{ji} !} \right) \right\} \right] \cdots \int_{0}^{\infty} \prod_{j=1}^{\sum s_{ji}} \left( \frac{\rho_{ji}}{n_{ji} !} \right) \, du_1 \cdots du_q \]
Now by the multinomial theorem,

\[ G = (\prod K_j)^{k_j - 1} \sum_{0}^{\infty} \sum_{0}^{\infty} \exp \left\{ - \sum_{i=1}^{q} u_i \right\} \prod_{j=1}^{q} \rho_{ji} u_{ii} + \sum_{i=q+1}^{s} \rho_{ji} \right\}^j_{j=1} du_1 \ldots du_q \] .. (24)

It is to be noted that the parameter \( \rho_{ji} \) for all the Type 3 centers appear lumped together. Hence we may simplify the notation by introducing \( \rho_{jo} \) where

\[ \rho_{jo} = \sum_{i=q+1}^{s} \rho_{ji}, \quad j = 1, 2, \ldots, p. \] .. (25)

Let \( I \) be the collection of indices of classes of the former type and let \( I^* \) be the complementary collection, i.e.

\[ j \in I \iff \rho_{jo} > 0 \quad \text{and} \quad j \in I^* \iff \rho_{jo} = 0. \] .. (26)

With this notation,

\[ G = \left( \prod_{j=1}^{k_j} \rho_{jo} / \prod_{j=1}^{k_j} k_j! \right) \sum_{0}^{\infty} \sum_{0}^{\infty} \exp \left\{ - \sum_{i=1}^{q} u_i \right\} \prod_{j=1}^{q} \rho_{ji} u_{ii} + \sum_{i=q+1}^{s} \rho_{ji} \right\}^j_{j=1} du_1 \ldots du_q \] .. (27)

In vector notation, which we shall use widely, this reduces to

\[ G = \left( \prod_{j=1}^{k_j} \rho_{jo} / \prod_{j=1}^{k_j} k_j! \right) \sum_{0}^{\infty} \sum_{0}^{\infty} \exp \left\{ - \sum_{i=1}^{q} u_i \right\} \prod_{j=1}^{q} \rho_{ji} u_{ii} + \sum_{i=q+1}^{s} \rho_{ji} \right\}^j_{j=1} du_1 \ldots du_q \] .. (28)

where \( u = (u_1, u_2, \ldots, u_q)' \) \( \hat{I} = (1, 1, \ldots, 1)' \) .. (29)
\[ r_j = (r_{j1}, r_{j2}, \ldots, r_{jq}), \quad 1 \leq j \leq p \]

\[ r_{ji} = \frac{\rho_{ji}}{\rho_{jo}} \text{ if } j \in I \]

\[ = -\rho_{ji} \quad \text{if } j \in I' \]

\[ \delta_{ji} = \begin{cases} 1 & \text{if } j \in I \\ 0 & \text{if } j \in I' \end{cases} \]

\[ Q^+ = \{ u | u \geq 0 \}. \]

We now introduce the large parameter \( N \) and define

\[ \beta_j = \frac{K_j}{N}, \quad 1 \leq j \leq p, \quad \cdots \tag{30} \]

\[ \Gamma_j = N r_j, \quad 1 \leq j \leq p. \quad \cdots \tag{31} \]

In practice we have used

\[ N = \max_{i,j} \left\{ \frac{1}{r_{ji}} \right\}. \quad \cdots \tag{32} \]

On substituting (30) and (31) into (28) and after the change of variables \( z = u/N \), we obtain

\[ G(k) = \left[ N^q \prod_{j \in I} \frac{\rho_{jo}}{K_j!} \right] \int_{Q^+} e^{Nf(z)} \, dz \]

**Result 6.2.2**

For class index \( \sigma, \quad 1 \leq \sigma \leq p, \) and center index \( i, \quad 1 \leq i \leq q, \)

\[ u_{oi} (K+e_{o})^{-1} = \left\{ \frac{1}{r_{oi} (K_{o}+1)} \right\} \left\{ \delta_{oi} + \frac{\left\{ Q_+ (I_{o}^\sigma z) e^{Nf(z)} \, dz \right\}}{Q_+ e^{Nf(z)} \, dz} \right\} \]
Proof

Consider the mean value $u^i_\sigma (K)$ which gives the utilization of the $i^{th}$ processor by jobs of the $\sigma^{th}$ class for a population distribution by class in the network denoted by $K = (K_1, K_2, \ldots, K_p)$. Other mean performance indices such as throughput and mean response time are well known to be simply related to $\{u^i_\sigma (K)\}$.

Bruell [8] established that

$$u^i_\sigma (K) = \frac{G(K - e_\sigma)}{G(K)}, \quad 1 \leq \sigma \leq p, \quad 1 \leq i \leq q. \quad \ldots (33)$$

where $e_\sigma$ is our notation for the vector with the $\sigma^{th}$ component unity and all other components zero. Thus the value for the partition function is needed for the given population distribution and also for the population in the $\sigma^{th}$ class reduced by 1.

Now from (19)

$$G(K + e_\sigma) = \frac{\rho_\sigma \prod_{j \in \Omega} k_j}{(K_\sigma + 1)! K_j!} \int_{Q^+} e^{\nu u} (1 + r^\nu u) \prod_j (\delta_{ji} + r_j^\nu u)^{k_j} du \quad \text{if } \sigma \in \Omega. \quad \ldots (34)$$

$$= \frac{\prod_{j \in \Omega} k_j}{(K_\sigma + 1)! K_j!} \int_{Q^+} e^{\nu u} (r^\nu u) \prod_j (\delta_{ji} + r_j^\nu u)^{k_j} du \quad \text{if } \sigma \in \Omega^*. \quad \ldots (35)$$

From (33)-(35) and the same change of variables, namely $z = u/N$, employed in transforming (19) to (20) we obtain

$$u^i_\sigma (K + e_\sigma)^{-1} = \left\{ \frac{1}{r^i_\sigma (K_\sigma + 1)} \right\} \left\{ \delta^i_\sigma + \frac{\int_{Q^+} (\Gamma^\nu z) e^{N(z)} dz}{\int_{Q^+} e^{N(z)} dz} \right\}$$
We henceforth consider only networks in which the route for each class always contains an infinite server center. Specifically,

\[ \rho_{jo} > 0, \quad j = 1, 2, \ldots, p \]

and the set \( I^* \) is empty.

Define \( \alpha = 1 - \sum \beta_j \Gamma_j \)

so that in terms of the original network parameters

\[ \alpha_i = 1 - \sum K_j \frac{\rho_{ji}}{\rho_{jo}}, \quad i = 1, 2, \ldots, q \]

It is important to note that \( \alpha \) is independent of the choice of \( N \). Normal usage in large networks will almost certainly require \( \alpha_i > 0 \) and in all likelihood \( \alpha_i \) will not be close to 0 for all \( i \). We assume \( \alpha_i > 0, \quad i = 1, 2, \ldots, q \)

which condition we refer to as “normal usage”.

We observe that

\[ f(0) = 0 \]

and

\[ \nabla f(z) = 1 - \sum_j \frac{\beta_j}{1 + \Gamma_j' z} \Gamma_j \]

so that

\[ \alpha = \nabla f(0). \]

The assumption of \( \alpha > 0 \) and the form in (36) ensures that the function \( f \) has no stationary points in \( Q^+ \) since

\[ \nabla f(z) \geq \nabla f(0) > 0, \quad z \in Q^+. \]

Also we observe that

\[ \left\{ \frac{\partial^2 f}{\partial z_{i_1} \partial z_{i_2}} \right\} = \sum \frac{\beta_j}{(1 + \Gamma_j' z)^2} \Gamma_j \Gamma_j' \]
from which we note that the Hessian is positive semi-definite.

To conclude, with "normal usage" f is a convex function with its minimum in $Q^+$ attained at 0 and with no point in $Q^+$ where its gradient vanishes.

Consider the following transformations on the basic integral.

$$
\int_{Q^+} e^{-Nf(z)} \, dz = \int_{Q^+} e^{-Nf(z)+N \Sigma \beta_j \{ \Gamma_j \cdot z \} - N \Sigma \beta_j \{ \Gamma_j \cdot z \} } \, dz
$$

$$
= \int_{Q^+} e^{-N\alpha^T \cdot \mathbf{z}} \exp\left\{N \sum_j \beta_j \{ \Gamma_j \cdot z \} \right\} \, dz
$$

$$
= N^{-q} \int_{Q^+} e^{-\alpha^T \cdot \mathbf{v}} \exp\left\{-\sum_j \beta_j \{ \Gamma_j \cdot \mathbf{v} - N \log \left[ 1 + \frac{1}{N} \right] \} \right\} \, d\mathbf{v}
$$

(37)

where $\mathbf{v} = \alpha \cdot \mathbf{z}$. Now make the following change of variables.

$$
\nu_i \triangleq \alpha_i \cdot u_i, \quad 1 \leq i \leq q
$$

and normalize the system parameters with respect to $\alpha$ thus,

$$
\hat{\Gamma}_{ji} \triangleq \frac{\Gamma_{ji}}{\alpha_i}, \quad 1 \leq j \leq p, \quad 1 \leq i \leq q.
$$

We observe that in particular

$$
\Gamma_j \cdot \mathbf{u} = \hat{\Gamma}_j \cdot \mathbf{v}.
$$

From (37)

$$
\int_{Q^+} e^{-Nf(z)} \, dz = \frac{N^{-q}}{(\Pi \alpha_i)} \int_{Q^+} e^{-\hat{\beta}^T \cdot \mathbf{v}} H(N^{-1}, \mathbf{v}) \, d\mathbf{v}
$$

where

$$
H(N^{-1}, \mathbf{v}) \triangleq e^{s(N^{-1}, \mathbf{v})}
$$

$$
s(N^{-1}, \mathbf{v}) \triangleq -\sum_{j=1}^{p} \beta_j \{ \hat{\Gamma}_j \cdot \mathbf{v} - N \log \left[ 1 + \frac{1}{\hat{\Gamma}_j \cdot \mathbf{v}} \right] \}
$$

(38)

Let

$$
I(N) \triangleq \int_{Q^+} e^{-\hat{\beta}^T \cdot \mathbf{v}} H(N^{-1}, \mathbf{v}) \, d\mathbf{v}
$$
and
\[ I_{(1)}(N) = \int_{Q^+} e^{\hat{v}v} (\Gamma^* \sigma v) H(N^{-1}, v) \, dv \]

where the superscript (1) is the mnemonic for first moment.

**Result 6.2.3**

\[ I(N) \sim \sum_{k=0}^{\infty} \frac{A_k}{N^k} \quad \ldots (39) \]

where \( A_k = \int_{Q^+} e^{\hat{v}v} h_k(v) \, dv \), \( \{h_k(v)\} \) is obtained recursively from (48) with leading elements exhibited in (49), and \( \{f_k(v)\} \) is as in (44).

**Proof**

We let
\[ H(N^{-1}, v) = \sum_{k=0}^{\infty} \frac{h_k(v)}{N^k} \quad \ldots (40) \]

and
\[ A_k = \int_{Q^+} e^{\hat{v}v} h_k(v) \, dv \quad \ldots (41) \]

and claim that
\[ I(N) \sim \sum_{k=0}^{\infty} \frac{A_k}{N^k} \quad \ldots (42) \]

Let us elaborate on the coefficients \( \{h_k(v)\} \) in (40)

\[ h_k(v) = \frac{1}{k!} \left. \frac{\partial^k}{\partial(1/N)^k} H(0, v) \right|_{k=0, 1, 2, \ldots} \quad \ldots (43) \]
To make these explicit we need to first introduce

\[ f_k(v) = \frac{(-1)^k}{k} \sum_j \beta_j (\Gamma_j v)^k, \ k = 1, 2, ... \]  

(44)

Their role becomes clear if

\[ H(N^{-1}, v) = e^v (N^{-1}, v) \]  

(45)

and note from (38) that for fixed \( v \in Q^+ \), \( s(N^{-1}, v) \) and hence \( H(N^{-1}, v) \) are functions of \( N^{-1} \), analytic in \( \text{Re} (N^{-1}) > \epsilon(v) \), where \( \epsilon(v) < 0 \). Then

\[ s^{(k)}(0, v) = -k! f_{k+1}(v), \ k = 1, 2, ... \]  

(46)

To proceed now to the derivatives of \( H(\cdot, v) \) itself, we will find useful the following expression in which it is understood that all derivatives are with respect to \( N^{-1} \):

\[ H^{(k+1)}(N^{-1}, v) = \sum_{m=0}^{k} \binom{k}{m} s^{(k+1-m)}(N^{-1}, v) H^{(m)}(N^{-1}, v), \ k = 0, 1, ... \]  

(47)

From (43), (46) and the above it is easy to see that there exists the following recursive scheme for generating \( \{h_k(v)\} \):

\[ h_0(v) = 1 \]

\[ h_{k+1}(v) = -\frac{1}{k+1} \sum_{m=0}^{k} (k+1-m) f_{k+2-m}(v) h_m(v), \ k = 0, 1, 2, ... \]  

(48)

In particular the leading elements are

\[ h_k(v) = 1, \ k = 0 \]  

\[ = -f_2(v), \ k = 1 \]  

(49)
Thus the theorem follows.

SECTION 6.3. JACKSON NETWORKS

M/G/1 Processor-sharing queueing system

This model is shown in Fig.6.3.1. Customers were assumed to arrive at the processor in a Poisson stream. Each arriving customer entered the single queue and waited in a FCFS fashion for a quantum $\Delta s$ of CPU service. When the quantum of service was used up, if the customer required more service, a return was made to the tail of the same queue to repeat the cycle. The cycle was repeated by the customer until the required service was received, whereupon the customer departed. Kleinrock [33] discovered that by letting the quantum $\Delta s$ shrink to zero he got an analytical model with much simpler expressions for the performance measures but which was a good approximation to a round-robin system with a finite quantum. Such a system is called a processor-sharing system since the customers are sharing the processor equally. We now give a formal description of the M/G/1 processor-sharing model.
The M/G/1 processor-sharing queueing system has a Poisson arrival pattern with average arrival rate $\lambda$. The service time distribution is general with average rate $\mu$. The queue discipline is processor-sharing, which means each arriving customer immediately starts receiving his share of service, so there is no queue. Thus, if a customer arrives when there are already $n-1$ customers in the system, the customer receives service at the average rate $\mu/n$. The following steady state relations hold, when $\rho = \lambda/\mu < 1$:

$$p_n = P[N = n] = (1 - \rho) \rho^n, \quad n = 0, 1, 2, \ldots$$  \hspace{1cm} \ldots (50)

$$L = E[N] = \rho / (1 - \rho),$$  \hspace{1cm} \ldots (51)

$$E[w|s = t] = t(1 - \rho),$$  \hspace{1cm} \ldots (52)

$$W = E[w] = E[s] / (1 - \rho).$$  \hspace{1cm} \ldots (53)

Although there is no queue, a customer requiring $t$ units of service time suffers a delay because the full capacity of the CPU is not available. The average of this delay, which we denote by $E[q|s = t]$, is the difference between $E[w|s = t]$ and $t$, the amount of CPU time needed. Thus we have the following:
\[
E[q|s = t] = \rho t / (1-\rho), \quad \ldots (54)
\]
\[
E[q] = \rho E[s]/(1-\rho). \quad \ldots (55)
\]

Here \(E[q]\) is the average delay experienced by customers.

These equations are exactly the same as the corresponding equations for the M/M/1 queueing system. In addition the departure stream is Poisson. The distribution of \(w\) and \(q\) is not known in general. However, for the special case of exponential service time, Coffman et al. [11] have derived the Laplace transform of the conditional delay and its variance.

Equation (52) shows that the mean conditional response time is a linear function of required service time. A customer who requires twice as much service time as another will, on the average, spend twice as much time in the system. The mean response time \(W\), by (53), is independent of the service time distribution and depends only upon its mean value.

**Illustration 6.3.1**

A number of active terminals feed a Poisson stream of requests to a computer which operates approximately in processor-sharing mode with an average processing rate of 500,000 instructions/second. The average arrival rate \(\lambda = 4.77\) customers/second and the average number of instructions per customer is 100,000. We shall find the average response time \(W\) and the average number of customers being processed by the CPU, \(L = E[N]\). Also we shall calculate the probability that there are 5 or more customers being processed by the computer.
Solution

\[ E[s] = \frac{100,000}{500,000} = 0.2 \text{ second.} \]

Thus

\[ \rho = \lambda E[s] = 4.77 \times 0.2 = 0.954 \]

Hence, by (53),

\[ W = \frac{E[s]}{1 - \rho} = 4.348 \text{ seconds}. \]

By (51),

\[ L = E[N] = \frac{\rho}{1 - \rho} = 20.74. \]

With the geometric distribution of (50) we have

\[ P[N \geq n] = \rho^n. \]

Hence,

\[ P[N \geq 5] = (0.954)^5 = 0.79. \]

Theorem 6.3.1. (Jackson's Theorem)

Suppose a queueing network consists of \( m \) nodes satisfying the following three conditions.

(i) Each node has identical exponential servers. Node \( i \) has \( c_i \) servers each of whom has average service time \( 1/\mu_i \).

(ii) Customers arriving at node \( i \) from outside the system arrive in a Poisson pattern with average rate \( \lambda_i \). (Customers also arrive at node \( i \) from other nodes within the network).

(iii) Once served at node \( i \), a customer goes (instantly) to node \( j \) (\( j = 1,2,\ldots,m \)) with probability \( p_{ij} \); or leaves the network with probability

\[ 1 - \sum_{j=1}^{m} p_{ij}. \]
For each node \( i (i = 1, 2, \ldots, m) \) it can be seen that the average arrival rate to the node, \( \lambda_i \), is given by

\[
\lambda_i = \lambda_i + \sum_{j=1}^{m} p_{ji} \lambda_j.
\]

Then, if we let \( p (k_1, k_2, \ldots, k_m) \) denote the steady state probability that there are \( k_i \) customers in the \( i^{th} \) node \( (i = 1, 2, \ldots, m) \), and, if \( \lambda_i < c_i \mu_i \) for all \( i \), it is true that

\[
p(k_1, k_2, \ldots, k_m) = p_1(k_1) p_2(k_2) \cdots p_m(k_m),
\]

where \( p_i(k_i) \) is the steady state probability there are \( k_i \) customers in the \( i^{th} \) node if it is treated as an M/M/c_i queueing system with average arrival rate \( \lambda_i \) and average service time \( 1/\mu_i \) for each of the \( c_i \) servers. Furthermore, each node \( i \) behaves as if it were an independent M/M/c_i queueing system with average Poisson arrival rate \( \lambda_i \).

**Two-stage cyclic queueing models**

A number of queueing theory models have been used to study multiprogrammed computer systems. In such a system several programs are stored simultaneously in the main memory. Each program consists of a sequence of CPU and I/O instructions. While an I/O unit is processing some input or output for one program, which cannot execute any more CPU instructions until the I/O is completed, the CPU processes another program. The execution of a program in such a system is characterized by its cyclical movement between the CPU, and the I/O units until execution is completed; the program then leaves the system. Thus a program in main memory is in one of four states; waiting for the CPU, in CPU execution, waiting for I/O, or in I/O execution. Hence such a system can be modeled by the two-stage cyclic queueing network shown in Fig.6.3.2.
Fig. 6.3.2. Two-stage cyclic queueing model of multiprogramming

The model shown in Fig. 6.3.2. can also be used to study a computer having virtual memory operating under demand paging. In such a system both main memory and programs are partitioned into pieces of equal size called pages. Each program in main memory is assigned a fixed number of pages of memory. If a program in CPU execution references a page not in main memory, the missing page is brought in from secondary memory to replace a page in main memory. While this paging activity occurs, the CPU processes another program. Eventually the original program returns to CPU execution until it experiences another page fault. Hence, the cyclic queueing network of Fig. 6.3.2 can be used to study this multiprogramming model, also, with paging corresponding to the I/O activity.

In this cyclic queueing model we assume a Poisson arrival process with average arrival rate $\lambda$. It is also assumed that both the CPU and the I/O unit provide exponential service with average service times $1/\mu_1$ and $1/\mu_2$, respectively. An arriving program queues for service at the CPU. When a CPU service is completed, the program either leaves the system (with probability $p$) or queues for an
I/O service (with probability 1-p). When the I/O service is over, the program rejoins the CPU queue. This cycle is repeated until the program completes execution. In this model the queues are assumed to have infinite capacity; there is no blocking due to lack of space in a queue.

We can calculate the steady state statistics of this model by using Jackson's theorem. Let $\lambda_1$ and $\lambda_2$ be the average arrival rates to the CPU and I/O unit, respectively. Then we have

$$\lambda_1 = \frac{\lambda}{p}.$$  

Clearly

$$\lambda_2 = (1 - p) \lambda_1 = (1 - p) \frac{\lambda}{p}.$$  

The respective server utilizations for the CPU and I/O units are calculated by

$$\rho_1 = \frac{\lambda_1}{\mu_1} = \frac{\lambda}{p \mu_1},$$  

and

$$\rho_2 = \frac{\lambda_2}{\mu_2} = (1 - p) \frac{\lambda}{p \mu_2}.$$  

The average throughput $\lambda_T$, measured in jobs per unit time, is given by

$$\lambda_T = p \lambda_1 = \lambda.$$  

Other two-stage cyclic queueing models

One characteristic of most multiprogramming computer systems is that the number of programs kept in main memory is constant; whenever a program completes execution, another replaces it in main memory. Now consider the model shown in Fig. 6.3.2 but with the additional condition that the queueing network always contain exactly K customers (jobs).
This means that the incoming traffic must be sufficiently heavy that, whenever a job completes its service and departs, another is ready to take its place. The arrival pattern into the queueing network can no longer be regarded as Poisson; however it is an infinite arrival pattern.

The distribution of jobs in the system is completely specified by giving the number $n$ in queue or service at the CPU, for then $K - n$ are in queue or service at the I/O unit. Let $p_n$ be the probability there are $n$ jobs queued for or in service at the CPU.

Mitrani [42] shows that the following formulae are true:

$$p_n = \left\{ \frac{1 - r}{1 - r^{K+1}} \right\} r^n, \ n = 0, 1, 2, \ldots, K.$$ .... (56)

where $r = \mu_2/(1-p)\mu_1$.

The server utilizations for the CPU and I/O unit are given by

$$\rho_1 = 1 - p_0 = (1 - r^{k+1}) / (1 - r^{k+1}),$$ .... (57)

and

$$\rho_2 = 1 - p_k = (r - r^k) / (1 - r^{k+1}),$$ .... (58)

respectively.

The average response time is given by

$$K/\mu_1 \ p \ \rho_1.$$ .... (59)
To get the throughput $\lambda_T$, which is the average rate at which jobs depart the system, we reason that the departure rate is $\mu_1 p$ when the CPU is busy and zero otherwise, so

$$\lambda_T = \mu_1 p(1 - p_o) + 0 p_o = \mu_1 p \rho_1.$$

In Algorithm H, we show how to construct a two-stage hyperexponential distribution with a given mean and squared coefficient of variation, provided the latter is greater than or equal to one.

**Algorithm 6.3.1 (Algorithm H)**

Given $C^2 \geq 1$ and $\mu > 0$, this algorithm will produce the parameters for a two-stage hyperexponential random variable $X$ with squared coefficient of variation $C_X^2 = C^2$ and mean $E[X] = 1/\mu$. The distribution function of $X$ will be given by

$$F(x) = 1 - \alpha_1 e^{\mu_1 x} - \alpha_2 e^{\mu_2 x}, \quad x > 0.$$  .... (60)

**Step 1** [Calculate $\alpha_1$ and $\alpha_2$] Set $\alpha_1 = \frac{1}{2} \left\{ 1 - [(C^2 - 1)/(C^2 + 1)]^{1/2} \right\}$ and $\alpha_2 = 1 - \alpha_1$.

**Step 2** [Calculate $\mu_1$ and $\mu_2$] Set $\mu_1 = 2 \alpha_1 \mu$ and $\mu_2 = 2 \alpha_2 \mu$.

**Step 3** [Produce $F$] The distribution $F(x)$, defined by (60) with parameters calculated in Step 1 and Step 2, is the distribution function of a two-stage hyperexponential random variable having the required properties.
The central server model of multiprogramming

K Circulating markers (Programs)

![Diagram of the central server model of multiprogramming](image)

Fig. 6.3.3. Central server model of multiprogramming

This model, shown in Fig.6.3.3, is a closed model since it contains a fixed number of programs which can be thought of as markers that cycle around the system interminably. However, each time a marker (program) makes the cycle from the CPU directly back to the CPU we assume a program execution has been completed and a new program enters the system. There are M-1 I/O devices, each with its own queue, and each exponentially distributed with average service rate $\mu_i$ (i = 2, 3, ..., M). The CPU also is assumed to provide exponential service (with average rate $\mu_1$).

Upon completion of a CPU interval the job returns to the CPU (completes execution) with probability $p_1$ or requires service at I/O device i with probability $p_i$, i = 2, 3, ..., M. Upon completion of I/O service the job returns to the CPU queue for another cycle. If we let $k = (k_1, k_2, ..., k_M)$ represent the state of the system, where $k_i$ is the number of jobs (markers) at the $i^{th}$ queue (queueing or in
service) then Buzen [10, 9] shows the probability \( p(k_1, k_2, \ldots, k_M) \) that the system is in state \( k \) is given by

\[
p(k_1, k_2, \ldots, k_M) = \frac{1}{G(K)} \prod_{i=2}^{M} \left( \frac{\mu_i p_i}{\mu_i} \right)^{k_i}
\]

where \( G(K) \) is defined so as to make the probabilities sum to 1. Thus

\[
G(K) = \sum_{k \in S(K,M)} \prod_{i=2}^{M} \left( \frac{\mu_i p_i}{\mu_i} \right)^{k_i}
\]

where

\[
S(K, M) = \{(k_1, k_2, \ldots, k_M) \mid \sum_{i=1}^{M} k_i = K \text{ and each } k_i \geq 0\}.
\]

It can be shown that there are

\[
\begin{pmatrix}
M + K - 1 \\
-1
\end{pmatrix}
= \begin{pmatrix}
M + K - 1 \\
K
\end{pmatrix}
\]

elements in \( S(K, M) \).

We will show in the following Algorithm, the technique developed by Buzen for calculating \( G(0) = 1, G(1), G(2), \ldots, G(K) \).

**Algorithm 6.3.2 (Buzen's Algorithm)**

Given the parameters of the central server model of Fig. 6.3.3 (that is, \( \mu_i, p_i \) for \( i = 1, 2, \ldots, M \)), this algorithm will generate

\( G(K), G(K - 1), G(K - 2), \ldots, G(1), G(0) = 1 \).

**Step 1** [Assign values to the \( x_i \)] Set \( x_1 = 1 \) and then set \( x_i = \mu_i p_i / \mu_i \) for \( i = 2, 3, \ldots, M \).

**Step 2** [Set initial values] Set \( g(k, 1) = 1 \) for \( k = 0, 1, \ldots, K \) and set \( g(0, m) = 1 \) for \( m = 1, 2, \ldots, M \).
Step 3 [Initialize k] Set k to 1.
Step 4 [Calculate $k^{th}$ row] Set
\[ g(k, m) = g(k, m-1) + x_m g(k-1, m), \quad m = 2, 3, \ldots, M. \]
Step 5 [Increase k] Set k to k+1.
Step 6 [Algorithm complete] If $k \leq K$ return to Step 4.
Otherwise terminate the algorithm. Then $g(n, M) = G(n)$ for $n = 0, 1, \ldots, K$.

The traditional treatment of Jackson networks (and product form networks in general) uses the ideas of partial and detailed balance, quasi-reversible queues, etc. Jackson networks may be viewed as QBDs as we show here.

An example of an open Jackson network is given in Figure 6.3.4. These are networks of exponential servers with infinite buffers, fed from the outside by Poisson processes. At the end of a service at station i, a customer may decide to leave the
system with probability $P_{i,j}$, or to enter the queue in front of server $j$ with the probability $P_{i,j}$. In this particular example, there are three stations, two Poisson arrival processes to the stations 1 and 2 but no direct arrival to station 3; upon completion of their service at station 1, the customers move to station 3 with probability $P_{1,3} = 1$; upon completion of their service at station 2, the customers may either leave the system with probability $P_{2,2}$, or move to station 3 with probability $P_{2,3}$; upon completion of their service at station 3, customers leave the system with probability $P_3$, or feed back to station 1 with probability $P_{3,1}$.

This is a Markov process on a three-dimensional state space, infinite in each dimension: $\{(n_1, n_2, n_3); n_1 \geq 0, n_2 \geq 0, n_3 \geq 0\}$. The transitions are given in Table 6.3.1.

**Table 6.3.1. Transitions for the Jackson network of Figure 6.3.4**

<table>
<thead>
<tr>
<th>From $(n_1, n_2, n_3)$</th>
<th>To $(n_1+1, n_2, n_3)$</th>
<th>Rate $\lambda_1$</th>
<th>For $n_1, n_2, n_3 \geq 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(n_1, n_2, n_3)$</td>
<td>$(n_1+1, n_2, n_3-1)$</td>
<td>$\mu_3 P_{3,1}$</td>
<td>For $n_1, n_2 \geq 0; n_3 \geq 1$</td>
</tr>
<tr>
<td>$(n_1, n_2, n_3)$</td>
<td>$(n_1-1, n_2, n_3+1)$</td>
<td>$\mu_1$</td>
<td>For $n_1 \geq 1; n_2, n_3 \geq 0$</td>
</tr>
<tr>
<td>$(n_1, n_2, n_3)$</td>
<td>$(n_1, n_2+1, n_3)$</td>
<td>$\lambda_2$</td>
<td>For $n_1, n_2, n_3 \geq 0$</td>
</tr>
<tr>
<td>$(n_1, n_2, n_3)$</td>
<td>$(n_1, n_2-1, n_3)$</td>
<td>$\mu_2 P_{2,2}$</td>
<td>For $n_1, n_3 \geq 0; n_2 \geq 1$</td>
</tr>
<tr>
<td>$(n_1, n_2, n_3)$</td>
<td>$(n_1, n_2-1, n_3+1)$</td>
<td>$\mu_2 P_{2,3}$</td>
<td>For $n_1, n_3 \geq 0; n_2 \geq 1$</td>
</tr>
<tr>
<td>$(n_1, n_2, n_3)$</td>
<td>$(n_1, n_2, n_3-1)$</td>
<td>$\mu_3 P_{3,3}$</td>
<td>For $n_1, n_2 \geq 0; n_3 \geq 1$</td>
</tr>
</tbody>
</table>
If we define the level \( \mathcal{L}(n) \) as the subset of states where there are \( n \) customers in the station 1, we find that \( \mathcal{L}(n) \) is two dimensional since the states in \( \mathcal{L}(n) \) are indexed by \( (n_2, n_3) \). Thus, the infinitesimal generator has the structure

\[
Q = \begin{pmatrix}
B_1 & B_0 & 0 & 0 & \ldots \\
B_2 & A_1 & A_0 & 0 & \ldots \\
0 & A_2 & A_1 & A_0 & \ldots \\
0 & 0 & A_2 & A_1 & \ldots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}
\]

where the blocks \( B_i \) and \( A_i, \ i = 0, 1, 2 \), are doubly infinite. One may verify that

\[
A_0 = \begin{pmatrix}
\tilde{A}_0 & 0 & 0 & \ldots \\
0 & \tilde{A}_0 & 0 & \ldots \\
0 & 0 & \tilde{A}_0 & \ldots \\
\vdots & \vdots & \vdots & \ddots
\end{pmatrix} = B_0,
\]

with

\[
\tilde{A}_0 = \begin{pmatrix}
\lambda_1 & \ldots \\
\mu_3 p_{3,1} & \lambda_1 & 0 & \ldots \\
\ldots & \ldots \\
\mu_3 p_{3,1} & \lambda_1 & \ldots \\
\ldots & \ldots \\
\end{pmatrix}
\]

that
\[
\begin{align*}
A_2 &= \begin{pmatrix}
\tilde{A}_2 & 0 & 0 & \ldots \\
0 & \tilde{A}_2 & 0 & \ldots \\
0 & 0 & \tilde{A}_2 & \ldots \\
& & & \\
& & & \\
& & & \\
& & & \\
\end{pmatrix} = B_2, \\
\tilde{A}_2 &= \begin{pmatrix}
& & \mu_1 & \ldots \\
& & & \mu_1 \\
& & & \ldots \\
& & & \ldots \\
\end{pmatrix}, \\
\tilde{A}_1 &= \begin{pmatrix}
\tilde{A}_{11} & \tilde{A}_{10} & 0 & \ldots \\
\tilde{A}_{12} & \tilde{A}_{11} & \tilde{A}_{10} & \ldots \\
0 & \tilde{A}_{12} & \tilde{A}_{11} & \ldots \\
& & & \ldots \\
& & & \ldots \\
\end{pmatrix}, \\
\tilde{A}_{10} &= \begin{pmatrix}
\lambda_2 & & \ldots \\
& \lambda_2 & & \ldots \\
& & \lambda_2 & \ldots \\
& & & \ldots \\
\end{pmatrix},
\end{align*}
\]
\[ \tilde{A}_{12} = \begin{pmatrix} \mu_2 P_2, & \mu_2 P_{2,3} & \ldots \\ \cdot & \mu_2 P_2, & \mu_2 P_{2,3} \\ \cdot & \cdot & \mu_2 P_2 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}, \]

\[ \tilde{A}_{11} = \begin{pmatrix} \cdot & \cdot & \cdot & \ldots \\ \mu_3 P_3, & -\mu_3 & \cdot & \ldots \\ \cdot & \mu_3 P_3, & -\mu_3 & \ldots \\ \cdot & \cdot & \cdot & \ldots \end{pmatrix} - (\lambda_1 + \lambda_2 + \mu_1 + \mu_2) I; \]

and that \( B = A_1 + \mu_1 I \).

With this example, we see that the notion of levels covers situations which are very different. Here, the phase covers the whole remainder of the queueing network.

**Jackson network and QBDs**

In the case of a Jackson network, where the level is the number \( n_1 \) of customers in the first queue and the phase is the vector \( r = (n_2, \ldots, n_k) \) of number of customers in the remainder of the network, the matrices \( B, A_0, A_1 \) and \( A_2 \) are as follows:

\[ A_0 (r, r) = \lambda_1, \]

\[ A_0 (r, r, e_j) = \mu_j P_{j;1} \text{ for } 2 \leq j \leq K, \]

\[ \ldots (61) \]
where \( e_j \) is a row vector with all components equal to 0, except for a 1 in position \( j \), so that the difference between the states \( \mathbf{r} \) and \( \mathbf{r} - e_j \) is that a customer has left the station \( j \). Naturally, we restrict the transitions to feasible states, so that the rate (61) is only defined if \( n_j \geq 1 \).

Similarly, we have that
\[
A_2 (\mathbf{r}, \mathbf{r}) = \mu_1 q_1, \\
A_2 (\mathbf{r}, \mathbf{r} + e_j) = \mu_j p_{ij} \text{ for } 2 \leq j \leq K,
\]
and
\[
A_1 (\mathbf{r}, \mathbf{r} - e_i) = \lambda_i, \\
A_1 (\mathbf{r}, \mathbf{r} - e_j) = \mu_j q_{ij}, \\
A_1 (\mathbf{r}, \mathbf{r} - e_j + e_i) = \mu_j p_{ji}
\]
for \( 2 \leq i, j \leq K \). We assume that \( p_{ii} = 0 \) for all \( i \). Finally, we have that \( B = A_1 + \mu_1 I \).

Now the matrix \( A_1 \) may also be written as follows:
\[
A_1 (\mathbf{r}, \mathbf{r} + e_i) = F(\mathbf{r}, \mathbf{r} + e_i), \\
A_1 (\mathbf{r}, \mathbf{r} - e_j) = F(\mathbf{r}, \mathbf{r} - e_j) - \mu_j p_{ji}, \\
A_1 (\mathbf{r}, \mathbf{r} - e_j + e_i) = F(\mathbf{r}, \mathbf{r} - e_j + e_i), \\
\text{and } A_1 (\mathbf{r}, \mathbf{r}) = F(\mathbf{r}, \mathbf{r}) - \lambda_1 + \mu_1,
\]
where \( F \) is the generator of the Jackson network obtained by removing the node 1: no customer arrives there, and those customers who would be directed there, coming from another node, are redirected so that they leave the system altogether.

In order to prove that \( n_1 \) is independent from \( \mathbf{r} \) in steady state, we need to find \( \eta \) and \( \theta \) which satisfy the conditions
\[
\theta (A_0 - \eta \mu_1 I) = 0,
\]
\[ \theta (B + \eta A_2) = 0, \]

or, equivalently,

\[ \theta A_0 - \eta \mu_1 \theta = 0, \quad \ldots \]  (62)

\[ \theta (F - \lambda_1 I + \eta A_2) = 0, \quad \ldots \]  (63)

This would be enough to prove the product form property, since by symmetry we also prove that \( n_2 \) is, in steady state, independent of \( (n_1, n_3, \ldots, n_k) \), etc. However, the product form is even more obvious since we directly prove below that the components of \( \theta \) resolve into factors over the nodes 2 to \( K \).

**Theorem 6.3.2**

We denote by \( P \) the matrix of transition probabilities among the nodes and by \( y \) the solution of the traffic equations

\[ y = \lambda + yP, \quad \ldots \]  (64)

where \( \lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k) \).

If \( y_i / \mu_i < 1 \) for all \( i \), then the conditions \( \theta (A_0 + \eta A_1 + n^2 A_2) = 0 \) and \( \theta (B + \eta A_2) = 0 \) are satisfied by the scalar

\[ \eta = y_i / \mu_i \quad \ldots \]  (65)

and the vector \( \theta \) with

\[ \theta_r = c \prod_{2 \leq i \leq K} (y_j / \mu_j)^{n_j}, \quad \ldots \]  (66)

where

\[ c = \prod_{2 \leq i \leq K} (1 - y_j / \mu_j). \]
Proof

The proof is by induction on \( K \). The theorem clearly holds for \( K = 1 \) since the network then reduces to a simple \( M/M/1 \) queue. Therefore, we may assume that the theorem has been proved for networks with \( K - 1 \) nodes.

Using (65), (66), the left-hand side of (62) is written as follows:

\[
\begin{align*}
\lambda_1 \theta_r + \sum_{2 \leq j \leq K} \mu_j P_{jj} \theta_{r+ej} &- \eta \mu_1 \theta_r \\
= (\lambda_1 + \sum_{2 \leq j \leq K} y_j P_{jj} y_1) \theta_r & \text{for all } r,
\end{align*}
\]

which is equal to zero by (64).

The left-hand side of (63) is seen to be equal to

\[
\begin{align*}
\{\theta(F - \lambda_1 I + \eta A_2)\} (r) &= \theta_r F' (r, r) + \sum_{2 \leq j \leq K} \theta_{r-ej} F' (r-ej, r) \\
+ \sum_{2 \leq j \leq K} \theta_r + e_i F' (r + e_i, r) + \sum_{2 \leq j \leq K} \sum_{2 \leq l \leq K} \theta_{r-ej+e_l} F' (r-ej+e_l, r) \\
+ \theta_r (y_1 - \lambda_1) - \sum_{2 \leq j \leq K} \theta_{r+ej} \mu_j P_{jj},
\end{align*}
\]

where

\[
\begin{align*}
F' (r, r) &= F(r, r) - y_1 \sum_{2 \leq j \leq N} P_{ij}, \\
F' (r-ej, r) &= F(r-ej, r) + y_1 P_{ij}, \\
F' (r + e_i, r) &= F(r + e_i, r),
\end{align*}
\]

and

\[
F' (r - ej + e_i, r) = F(r - ej + e_i, r).
\]
The last two terms in (67) add up to zero by the traffic equation; therefore, the system reduces to

\[ \theta (F - \lambda_1 I + \eta A_2) = \theta F'. \] .... (68)

Moreover, the matrix \( F' \) is the generator of a Jackson network obtained by removing the node 1 and replacing the arrival rates to the other nodes by

\[ \lambda'_j = \lambda_j + y_1 P_{ij} \] for \( 2 \leq j \leq K. \]

By the induction assumption, the right-hand side of (68) will be equal to zero if \( (y_2, ..., y_k) \) is a solution of the balance equations for the reduced network, i.e., if

\[ y_j = \lambda'_j + \sum_{2 \leq i \leq K} y_i P_{ij} \]

\[ = \lambda_j + \sum_{i \leq K} y_i P_{ij} \] for \( 2 \leq j \leq K, \]

which holds by (64). Hence the theorem follows.

**SECTION 6.4. MEAN DELAY IN A QUEUEING NETWORK**

**Monte-Carlo connectivity analysis**

With large, irregular networks, our only recourse is to simulation. A useful model is one in which time is divided up into uniform intervals. During each interval a link (or IMP) is either up or down, but not partly up and partly down. The probability of a link failing during any interval is \( p. \)

There are several possible metrics that can be used for network reliability. The simplest is just whether or not the network is connected. For this metric the simulation output is the probability of disconnection as a function of \( p. \) Another useful metric is the fraction of IMPs that can still communicate, again as a function of \( p. \)
We can compute the fraction of IMPs unable to communicate for various values of $p$. It also computes the probability that the network becomes disconnected. The assumption is that only lines fail, never IMPs, although the same principle can be used for simulating IMP failures or mixed failures. After reading in the data specifying how many simulations run to make for each value of $p$, which values of $p$ to use, and the graph (in matrix form), the program performs one simulation run on the given graph with a given $p$. Each arc in the graph is inspected. If a random number whose probability density function is uniform between 0.0 and 1.0 is less than $p$, the arc is deleted.

After the arc detection phase is completed, we check to see which pairs of nodes are connected. It is to be noted that there are only $n(n-1)/2$ possible communicating pairs in an indirected graph. We do not count the diagonal elements.

**Queueing models in delay analysis**

In many design situations, there are two major constraints: reliability and delay. When network traffic is light, packet delay is primarily due to the time each IMP needs to store and forward the packet. If the distances are long, propagation delay may also play a role. However, as traffic increases, the principal delay becomes the queueing delay within each IMP, as packets must wait their turn to be sent out on a heavily used line. Consequently, our analysis of network delay will be based on a queueing theory model.
The network delay may be represented by the M/M/1 queueing model. The results derived for the M/M/1 queue can be applied to the problem of finding the queueing delay for packets in an IMP. First it is convenient to change the notation slightly to convert the units of service time from customers/sec to bits/sec. Let the probability density function for packet size in bits be \( \mu e^{-\mu x} \) with a mean of \( 1/\mu \) bits/packet. Let \( c_i \) be the capacity of the \( i^{th} \) communication channel than \( \mu c_i \) is the service rate in packets/sec. The arrival rate for the \( i^{th} \) channel is \( \lambda_i \) packets/sec.

Now the equation

\[
T = \frac{1}{\mu - \lambda},
\]

where \( T \) is the total waiting time, for the M/M/1 queueing model can be written for the \( i^{th} \) channel as

\[
T_i \approx \frac{1}{\mu c_i - \lambda_i}
\]

where \( T_i \) includes both queueing and transmission time.

To calculate the mean packet delay

For a network with \( n \) IMPs and \( m \) lines, it is convenient to define

\[
\gamma = \sum_{i=1}^{n} \gamma_i, \quad \text{and} \quad \lambda = \sum_{i=1}^{m} \lambda_i
\]

Although it might appear at first that summing the traffic between each pair of IMPs(\( \gamma \)) should give the same result as assuming the traffic on all the lines (\( \lambda \)) that is not so. The reason for this seeming anomaly is that some routes use two or more hops. If every route were three hops, then \( \lambda \) would be three times \( \gamma \). This ratio
$\bar{n} = \lambda / \gamma$ is the mean number of hops per packet. It is to be noted that $\bar{n}$ depends only on the relative traffic pattern and the routing algorithm, but not on the absolute volume of traffic. Doubling each element of $\gamma_{ij}$ has no effect on $\bar{n}$.

Further it is interesting to note that $\lambda$ can be related to the traffic matrix $(\gamma_{ij})$ by

$$\lambda = \frac{\sum_{j=1}^{n} \sum_{i=1}^{n} h_{ij} \gamma_{ij}}{\sum_{i=1}^{n} \sum_{j=1}^{n} h_{ij}}$$

where $h_{ij}$ is the number of hops in the route from IMP $i$ to IMP $j$. If shortest path routing is used, as it is in our example, the $\lambda$ so obtained is the theoretical minimum achievable for the given topology and traffic matrix.

The mean delay per hop is simply the sum of the individual line delays, from (69), weighted by the amount of traffic in the line, $\lambda_i$. To normalize the result, we divide by the total traffic, $\lambda$, yielding,

$$\text{mean delay per line} = \sum_{i=1}^{m} \frac{\lambda_i \; T_i}{\lambda}$$

However, the mean packet delay, $T$, is longer, because many packets must make several hops. Remembering that $\bar{n}$ is the mean number of hops per packet, we have

$$T = \frac{\bar{n} \sum_{i=1}^{m} \lambda_i \; T_i}{\lambda} = \bar{n} \sum_{i=1}^{m} \frac{\lambda_i / \lambda}{\mu c_i - \lambda_i} \quad \text{.... (70)}$$

Now we shall compute the mean packet delay when the network is almost empty. In an unloaded network, $\mu c_i \gg \lambda_i$ so we can neglect the term $\lambda_i$ in the denominator of equation (70), to get

$$T_0 = \frac{\bar{n} \sum_{i=1}^{n} \lambda_i / \lambda}{\mu c_i} \quad \text{.... (71)}$$
Illustration 6.4.1

Consider the cyclic queueing model shown in Figure 6.4.1. Assume that the lengths of successive CPU execution bursts are independent exponentially distributed random variables with mean $1/\mu$ and that successive I/O burst times are also independent exponentially distributed with mean $1/\lambda$.

At the end of a CPU burst a program requests an I/O operation with probability $0 \leq q_1 \leq 1$, and it completes execution with probability $q_0$ ($q_1 + q_0 = 1$). At the end of a program completion another statistically identical program enters the system, leaving the number of programs in the system at a constant level $n$ (known as the degree of multiprogramming).
Let the number of programs in the CPU queue including any being served at the CPU denote the state of the system, i, where $0 \leq i \leq n$. Then the state diagram is given by Figure 6.4.2. Let $\rho = \lambda / (\mu q_1)$. Then the steady-state probabilities are given by:

\[
p_i = \left\{ \begin{array}{ll}
\frac{\lambda}{\mu q_1} & p_0 = \rho^i \ p_0, \quad \text{and} \quad p_0 = \frac{1}{\sum_{i=0}^{n} \rho^i},
\end{array} \right.
\]

so that:

\[
p_0 = \left\{ \begin{array}{ll}
\frac{1 - \rho}{1 - \rho^{n+1}}, & \rho \neq 1, \\
\frac{1}{n+1}, & \rho = 1.
\end{array} \right.
\]

The CPU utilization is given by:

\[
U_0 = 1 - p_0 = \left\{ \begin{array}{ll}
\frac{\rho - \rho^{n+1}}{1 - \rho^{n+1}}, & \rho \neq 1, \\
\frac{n}{n+1}, & \rho = 1.
\end{array} \right.
\]

Fig. 6.4.2. The state diagram for the cyclic queueing model
Let $C(t)$ denote the number of jobs completed by time $t$. Then the (time) average $C(t)/t$ converges, under appropriate conditions, to a limit as $t$ approaches $\infty$. This limit is the average system throughput in the steady-state, and it is denoted here by $E[T]$. Whenever the CPU is busy, the rate at which CPU bursts are completed is $\mu$, and a fraction $q_o$ of these will contribute to the throughput. Then

$$E[T] = \mu q_o U_o.$$ 

For fixed values of $\mu$ and $q_o$, $E[T]$ is proportional to the CPU utilization.

Let the random variable $B_o$ denote the total CPU time requirement of a tagged program. Then $B_o = \text{EXP}(\mu, q_o)$. This is true because $B_o$ is the random sum of $K$ CPU service bursts, which are independent $\text{EXP}(\mu)$ random variables. Here the random variable $K$ is the number of visits to the CPU per program and hence is geometrically distributed with parameter $q_o$. The average number of visits $V_o$ to the CPU is $V_o = 1/q_o$, and thus $E[B_o] = V_o E[S_o] = 1/(\mu q_o)$, where $E[S_o] = 1/\mu$ is the average CPU time per burst.

The average throughput can now be rewritten as,

$$E[T] = \frac{U_o}{E[B_o]}$$

If $B_1$ represents the total I/O service time per program, then as in the case of CPU:

$$E[B_1] = \frac{q_1}{q_o} \frac{1}{\lambda} = V_1 E[S_1],$$
where the average number of visits $V_1$ to the I/O device is given by $V_1 = q_1/q_0$, and $E[S_i] = 1/\lambda$ is the average time per I/O operation. Now the parameter $\rho$ can be rewritten:

$$\rho = \frac{\lambda}{\mu q_1} = \frac{q_0 \lambda}{q_1} \cdot \frac{1}{E[B_o]} = \frac{E[B_o]}{E[B_i]}$$

Thus $\rho$ indicates the relative measure of the CPU versus I/O requirements of a program. If the CPU requirement $E[B_o]$ is less than the I/O requirement $E[B_i]$, that is, $\rho < 1$, the program is said to be I/O-bound; if $\rho > 1$, then program is said to be CPU-bound; and otherwise it is called balanced.

In Figure 6.4.3, we have plotted $U_o$ as a function of the balance factor $\rho$ and of the degree of multiprogramming $n$. When $\rho \ll 1$ or $\rho \gg 1$, $U_o$ is insensitive to $n$. Thus, multiprogramming is capable of improving throughput only when the workload is nearly balanced.
General method of calculating mean delay

We have considered the Computer network which can be modeled as M/M/1. We can generalize the discussion taking the consideration general arrival and general service. The waiting time distribution in the general case can be obtained as a solution of the integral equations. From that we can obtain the mean delay. In the neural network the situation is different, here we use fuzzy logic to classify the input and the output data sets broadly into different fuzzy classes by appropriate fuzzy membership function.

The analog of a neuron as a threshold element is shown in the following Fig. 6.4.4.

![Threshold element as an analog to a neuron](image)

**Fig.6.4.4. A threshold element as an analog to a neuron**

The variables $x_1, x_2, ..., x_i, ..., x_n$ are the n inputs to the threshold element. These are analogous to impulses arriving from several different neurons to one neuron. The variables $w_1, w_2, ..., w_i, ..., w_n$ are the weights associated with the impulses/inputs, signifying the relative importance that is associated with the path from which the inputs are coming.

When $w_i$ is positive, input $x_i$ acts as an excitatory signal for the element. When $w_i$ is negative, input $x_i$ acts as an inhibitory signal for the element.
The threshold element sums the product of these inputs and their associated weights \((\sum w_i x_i)\), compares it to a prescribed threshold value and, if the summation is greater than the threshold value, computes an output using a nonlinear function \(F\). The signal output \(y\) is a nonlinear function \(F\) of the difference between the preceding computed summation and the threshold value and is expressed as

\[ y = F(\sum w_i x_i - t) \]

where
- \(x_i\) = signal input \((i = 1, 2, \ldots, n)\)
- \(w_i\) = weight associated with the signal input \(x_i\)
- \(t\) = threshold level prescribed by user
- \(F(s)\) = a nonlinear function;

Popular choices for nonlinear function \(F(s)\) are a sigmoid function, a step function, and a ramp function.

Neural systems solve problems by adapting to the nature of the data they receive. To accomplish this is to use a training-data set and a checking-data set of input and output data/signals \((x, y)\). We start with a random assignment of weights \(w^{jk}_i\) to the paths joining the elements in the different layers. Then an input \(x\) from the training-data set is passed through the neural network. The neural network computes a value \((f(x)_{\text{output}})\), which is compared with the actual value \((f(x)_{\text{actual}} = y)\). The error measure \(\delta\) computed from these two output values is

\[ \delta = f(x)_{\text{actual}} - f(x)_{\text{output}} \]

This is the error measure associated with the last layer of the neural network. Next we try to distribute this error to the elements in the hidden layers using a technique called back-propagation.
The error measure associated with the different elements in the hidden layers is computed as follows. Let $\delta_j$ be the error associated with the $j^{th}$ element (Fig. 6.4.5).

Let $w_{nj}$ be the weight associated with the line from element $n$ to element $j$ and let $I$ be the input to unit $n$. The error for element $n$ is computed as

$$\delta_n = F'(I) w_{nj} \delta_j$$

The different weights $w_{jk}$ connecting different elements in the network are then corrected so that they can approximate the final output more closely. For updating the weights, the error measure on the elements is used to update the weights on the lines joining the elements.

For an element with an error $\delta$ associated with it, as shown in Fig. 6.4.6,
the associated weights may be updated as

$$w_i(\text{new}) = w_i(\text{old}) + \alpha \delta x_i$$

where $\alpha$ = learning constant

$\delta$ = associated error measure

$x_i$ = input to the element.

The input value $x_i$ is passed through the neural network again, and the errors, if any, are computed again. This technique is iterated until the error value of the final output is within some user-prescribed limits.

The neural network then uses the next set of input-output data. This method is continued for all data in the training-data set. This technique makes the neural network simulate the nonlinear relation between the input-output data sets. Finally a checking-data set is used to verify how well the neural network can simulate the nonlinear relationship.

The computer network can be identified as a neural network discussed above, where the delay in the computer network may be represented in terms of error discussed in the neural network case. For systems where we may have data sets of inputs and corresponding outputs, and where the relationship between the input and output may be highly nonlinear or not known at all, we may want to use fuzzy logic to classify the input and the output data sets broadly into different fuzzy classes.

We can derive a method by which fuzzy membership function may be created for fuzzy classes of input data set. We select a number of input data values and divide them into training-data set and checking-data set. The training-data set is
used to train the neural network. The data points are expressed with two coordinates and divided into different classes by clustering technique.

Initially the membership function is taken as unity and this proceeds to give a specification for second and third layer etc. A complete mapping of the membership of the different data point in different fuzzy classes can be derived to determine the overlap of different classes. Thus from the membership function, we can calculate the actual and output data, thereby deriving the delay.