52. Queuing Theory Applications to Communication Systems: Control of Traffic Flows and Load Balancing

With the tremendous increase in traffic on modern communication systems, such as the World Wide Web, it has made it imperative that users of these systems have some understanding not only of how they are fabricated but also how packets, which traverse the links, are scheduled to their hosts in an efficient and reliable manner. In this chapter, we investigate the role that modern queueing theory plays in achieving this aim. We also provide up-to-date and in-depth knowledge of how queueing techniques have been applied to areas such as prioritizing traffic flows, load balancing and congestion control on the modern internet system.

The Introduction gives a synopsis of the key topics of application covered in this chapter, i. e. congestion control using finite buffer queueing models, load balancing and how reliable transmission is achieved using various transmission control protocols.

In Sect. 52.1, we provide a brief review of the key concepts of queueing theory, including a discussion of the performance metrics, scheduling algorithms and traffic variables underlying simple queues. A discussion of the continuous-time Markov chain is also presented, linking it with the lack of memory property of the exponential random variable and with simple Markovian queues.

A class of queues, known as multiple-priority dual queues (MPDQ), is introduced and analyzed in Sect. 52.2. This type of queues consists of a dual queue and incorporates differentiated classes of customers in order to improve their quality of service. Firstly, MPDQs are simulated under different scenarios and their performance compared using a variety of performance metrics. Secondly, a full analysis of MPDQs is then given using continuous-time Markov chain. Finally, we show how the expected waiting times of different classes of customers are derived for a MPDQ.

Section 52.3 describes current approaches to assigning tasks to a distributed system. It highlights the limitations of many task-assignment policies, especially when task sizes have a heavy-tailed distribution. For these so called heavy-tailed workloads, several size-based load distribution policies are shown to perform much better than classical policies. Amongst these, the policies based on prioritizing traffic flows are shown to perform best of all.

Section 52.4 gives a detailed account of how the balance between maximizing throughput and congestion control is achieved in modern communication networks. This is mainly accomplished through the use of transmission control protocols and selective dropping of packets. It will be demonstrated that queueing theory is extensively applied in this area to model the phenomena of reliable transmission and congestion control.

The final section concludes with a brief discussion of further work in this area, an area which is growing at a rapid rate both in complexity and level of sophistication.

Queues, wherever they arise, are unfortunately an intrinsic part of human existence. We queue for items that are essential in our daily life as well as in situations that some would regard as an annoyance, although they are necessary, such as having to wait at a traffic intersection. On another level, modern communication systems, such as the internet, are under continuous strain in a world where it is not only demand for information that is increasing, but its speed of delivery. Given that some of this information has to be delivered over vast distances, is of varying sizes and is in competition for bandwidth with other traffic in the network, it has become essential in modern communication that traffic congestion be controlled, losses minimized and inefficient operations eradicated.

It has long been recognized that the problem of long delays suffered in many of our daily activities might be solved if one could model queues in all their manifestations. As a result, queueing theory was developed in the early part of the last century using tools and techniques from the well-established fields of probability and statistics to provide a systematic and general approach to understanding queueing phenomena. The earliest queueing applications are to problems of telephone congestion (pioneered by researchers such as *Erlang* [52.1] and *Palm* [52.2]). Subsequently, the subject was further developed and enriched with significant breakthroughs by researchers such as F. Pollaczek, A. Y. Khinchine, D. G. Kendall, D. R. Cox, J. R. Jackson, F. P. Kelly and many others. Queueing theory has been used to model many physical systems that involve delays, and currently, an important application is in modeling computer systems and communication networks.

This chapter considers the application of queueing theory to two critical issues of concern in modern communication systems, namely the problems of traffic flow control and load balancing, especially as they pertain to modern internet traffic. We begin in Sect. 52.1 with a brief review of basic queueing theory and then proceed to discuss the key topics of this paper in Sects. 52.2–52.4. The next three subsections provide a brief synopsis of these topics.

52.0.1 Congestion Control Using Finite-Buffer Queueing Models

Various scheduling algorithms have been introduced with the aim of improving quality of service (QoS) to customers. A wide variety of scheduling methods that aim to reduce congestion in communication systems have been studied. Many differentiate customers through marking and dropping processes (e.g. [52.3,4]). Others use time-marking and derivatives of this to allocate a degree of fairness in service, such as self-clocked fair queueing (SCFQ) and credit-based fair queueing (CBFQ) (e.g. [52.5,6]). A dual -queue length threshold (DQLT) [52.7] was used to divide real-time and non-real-time traffic to separate queues. Weighted round-robin (WRR) was looked at in [52.8].

In [52.9], a dual-queue (DQ) scheme was proposed to give better QoS to most customers at the expense of a few, rather than give poor QoS fairly to all customers. The dual-queue control scheme has two queues with finite space: namely the primary queue, which feeds into the service center, and the secondary queue, which acts as a waiting room when the primary queue is full (refer to Fig. 52.1). Upon arrival, a customer finding the primary

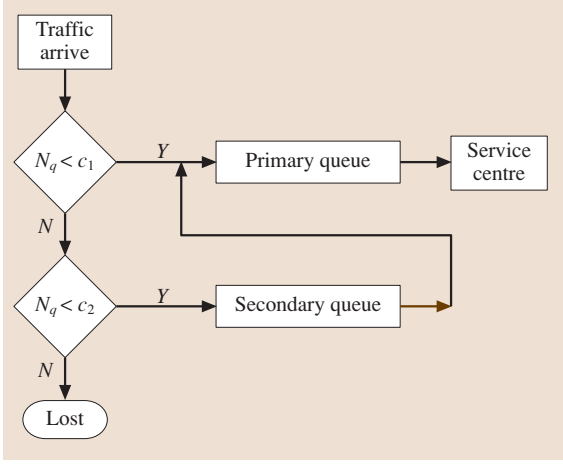

Fig. 52.1 The dual queue; N_q is the queue size, c_1 (c_2) is the primary (secondary) queue's buffer size

queue full waits in the secondary queue, if there is room. When a space becomes vacant in the primary queue, the customer at the front of the secondary queue joins the end of the primary queue. *Hayes* et al. [52.9] analyzed the delay characteristics of the dual queue against standard schemes such as first-come first-served (FCFS) and a modified deficit round-robin (DRR) technique [52.10], and demonstrated distinct advantages using the dual-queueing scheme. This work was extended to a wireless local area network [52.11] where minor modifications were made to the DQ and it was shown to outperform standard round-robin scheduling.

The multiple-priority dual queues (MPDQ), introduced in [52.12], builds upon this DQ scheme. The MPDQ introduces different classes into this scheme with the aim of providing better service to high-class customers without completely penalizing low-class ones. This is possible, not only because of the priority placed on customers' services, but also due to the MPDQ's partitioned queue structure. The MPDQ is especially relevant in the internet engineering task force (IETF) integrated services processes or differentiated services architecture. The MPDQ scheduling discipline provides a simple and effective mechanism for scheduling in these types of architecture.

52.0.2 Task Assignment Policy for Load Balancing

The usage of a *cluster* of commodity computers has become more prevalent in recent times. Such clusters are popular due to their scalable and cost-effective nature – often providing more computing resources at a significantly lower cost than traditional mainframes or supercomputers. They also provide other benefits, such as redundancy and increased reliability. The applications of such systems include supercomputing clusters and web-page serving for high-profile and high-volume websites, among other applications.

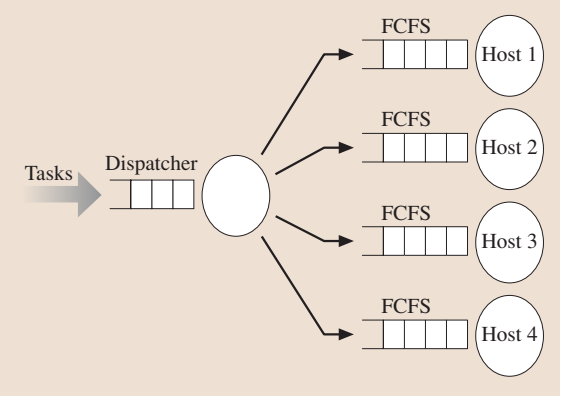

Fig. 52.2 Distributed server model

Figure 52.2 illustrates a common cluster configuration. Tasks, or *jobs* arrive at a central dispatcher, and are dispatched to hosts according to a *task assignment policy*. When a task arrives at the dispatcher, it is placed in a queue, waiting to be serviced in FCFS order.

The decision regarding which task assignment policy to utilize can significantly affect the perceived performance and server throughput. A poorly chosen policy could assign tasks to already overloaded servers, while leaving other servers idle, thus drastically reducing the performance of the distributed system. One major aim of a task assignment policy is to distribute tasks such that all available system resources are utilized and the load on the system is balanced. However, the ideal choice of task assignment policy is still an open question in many contexts.

The cluster configuration depicted in Fig. 52.2 is well suited to analysis via queuing theory. Equipped with some basic knowledge about our system of interest, such as the arrival and service distributions, we can easily obtain the expected performance metrics of the system. With these metrics, we can evaluate the performance of different task assignment policies, and make an informed judgment regarding which policy is best to employ.

52.0.3 Modeling TCP Traffic

The transmission control protocol (TCP) is a protocol that is widely used on the internet to provide reliable end-to-end connections. Reliability is achieved by verifying that each packet that enters the network is received correctly at the other end through the use of a return packet called an acknowledgment (ACK). It also provides congestion control, which attempts to prevent congestion collapse. Congestion collapse would occur if the amount of traffic entering the network greatly exceeded the capacity of the network. Congestion controls allows the amount of traffic entering the network to be controlled at the source.

The TCP congestion control mechanism uses a sliding window called a *congestion window* to control the rate at which packets are transmitted into the network. A congestion window has a size W measured in packets (actually its size is in bytes but it is simpler to think in terms of packets). This window is a segment of a larger buffer which starts at slot x and finishes at slot $x + W$. For example, a window of size $W = 3$ packets in a buffer

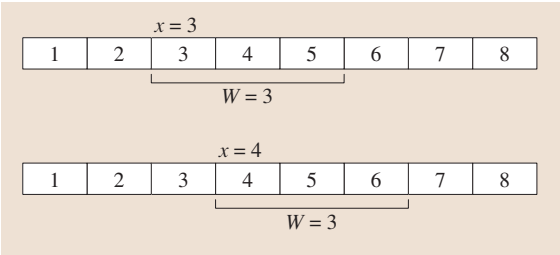

Fig. 52.3 Congestion window

of size eight slots could span slots either 1–3, 2–4, 3–5 and so on. Any packet within this window can be transmitted immediately into the network. When a packet is acknowledged by the receiving end, the window can slide across one place, allowing the next packet within the segment to be transmitted. For example if the window spans slot 3–5 then the window can slide to 4–6 and the packet can be transmitted in slot 6 when an ACK for packet 3 arrives (Fig. 52.3). This window size limits the maximum number of packets that can be in the network at any point in time.

The window size W changes over time as acknowledgements are received or lost. Queueing theory provides the ideal tool for analyzing the flow of packets within a network with TCP control, measuring its throughput and losses as packets are sent through a communication network.

52.1 Brief Review of Queueing Theory

The primary objective of queueing theory is to provide a systematic method of formulating, analyzing and predicting the behavior of queues. For example, customers waiting to be served at a store's checkout counter, cars waiting at an intersection controlled by traffic signals, telephone calls waiting to be answered and client request for an internet connection are a few of the countless phenomena that can be analyzed using standard queueing theory. The amount of literature devoted to queues and related problems is large and continues to grow at an exponential rate, especially since the advent of the World Wide Web. Published in 1961, the classic text on queueing theory, *Saaty* [52.13], has a list of over 900 papers in its bibliography. Since then, there have been many good texts on a broad range of queueing models, among which the following is a very short list of titles cited for their excellent coverage of key concepts and relevant examples: *Asmussen* [52.14], *Allen* [52.15], *Gross* and *Harris* [52.16], *Jaiswal* [52.17], *Kelly* [52.18] and *Kleinrock* [52.19, 20].

The basic queue is portrayed in Fig. 52.4. *Customers* arrive to the service facility from the environment and queue for service if there is someone ahead being serviced. After a length of time waiting, the customer is finally served, after which he departs from the queue. In this chapter, customers arrive as *single* units and not in *batches* or in *bulks*. Also, in the context of communication systems, the term *packets* will often be used interchangeably with *customers*. Each packet will be assumed to have a fixed uniform size; although in many systems packet sizes do vary, this does not affect the overall results unduly.

The representation in Fig. 52.4 leaves out much of the internal working of the service facility, neither does it describes how customers arrive into the system. Mathematically, the process can be described more precisely in the following way. Customers arrive from an *input source* requiring service. The n-th customer C_n arrives at time T_n, $n = 1, 2, \ldots$, where $0 < T_1 < T_2 < \ldots$. The inter-arrival times $\tau_n = T_{n+1} - T_n$, $n = 1, 2, \ldots$ are usually assumed to form a *renewal process*, i. e. they are independent and identically distributed random variables. Customer C_n requires a service time of duration s_n and these service times for different customers are also independent and identically distributed random variables. In addition, the processes $(\tau_n)_{n\geq 1}$ and $(s_n)_{n\geq 1}$ are also assumed to be statistically independent of each other.

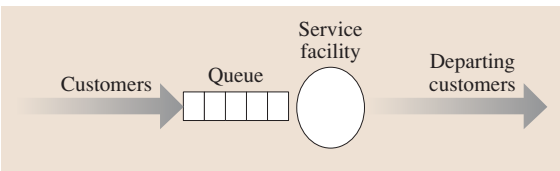

Fig. 52.4 The basic queue

52.1.1 Queue Characteristics

The characteristics that determine a basic queues are

- A: arrival pattern of customers;
- B: service pattern of customers;
- C: the number of servers;

- D: system capacity or buffer size;
- E: service discipline.

The representation of a queue using the notation $(A/B/C/D)$ with D often omitted is due to David Kendall and is known as Kendall's notation.

Arrival Pattern

This is the distribution of the renewal sequence $(\tau_n)_{n\geq 1}$. Some common distributions are

$$\text{Exponential M:}$$
$$f(t) = \lambda\, e^{-\lambda t}\,,\ t \geq 0\,,\ \lambda > 0\,.$$
$$\text{Deterministic } D$$
$$\text{Erlangian with } k \text{ stages}$$
$$E_k : f_k(t) = \frac{\lambda(\lambda t)^{k-1}\, e^{-\lambda t}}{(k-1)!}\,,\ t \geq 0\,, k = 1, 2, \ldots\,.$$

If the exact distribution is unspecified, $A = GI$ for *general and independent*. Note that the symbol M in the case of the exponential distribution stands for Markov due to the fact that certain stochastic processes in queues, where either the inter-arrival distribution or service distribution are exponentially distributed, have the Markov property (cf. Sect. 52.1.4). Markovian queues are analytically more tractable than other general types of queues. We note also that arrivals to a queue could occur in bulks with the size of each bulk fixed or distributed as a random variable. Bulk and related queues are extensively covered in the book by *Chaudhry* and *Templeton* [52.21].

Service Pattern

This is the distribution of the sequence $(s_n)_{n\geq 1}$. In the internet system, the distribution is also called the *task size distribution* since it refers to the distribution of the sizes of files that are requested by random arrivals of requests to the internet. The most common distributions are the same ones given for the arrival pattern. However, for many communication systems such as the World Wide Web, so called heavy-tailed distributions are more appropriate for file sizes than the exponential distribution (cf. Sect. 52.2). Similar to the arrival pattern, service could also be implemented in bulk, i. e. instead of customers being served as single units, they are served in bulks of fixed or arbitrary sizes.

System Capacity

Traditional queueing theory assumes there is no limit on the number of customers allowed into the service facility. Obviously this is an approximation of what occurs in reality where there *is a limit* to the number allowed into *any* system. In communication networks for example, buffer sizes associated with links between service nodes are finite and packets arriving to a saturated link are *dropped*. This dropping of packets is very commonly employed in modern active queue management (cf. Sect. 52.4).

Number of Servers

The service facility is manned by one, or often by more than one, server who provide service to customers. In communication networks where there are no visible servers, the servers are replaced by the notion of *bandwidth*, which refers to the portion of a transmission link (in megabytes/second, MBps) that a class of packets is allocated. The larger the bandwidth allocated, the faster these packets will traverse the link.

Service Disciplines

Customers are selected for service by a variety of rules called *service disciplines* or *scheduling algorithms*. The most common service discipline is *first-come first-served* (FCFS) where the customer who arrives first is served first. However, many other types of scheduling algorithms are used in modern communication such as last-in first-out (LIFO), SIRO (service in random order) and prioritized service. A *priority queue discipline* is one where the servers specify certain rules for serving customers according to their classes [52.17]. This is usually implemented with *preemption* and *non-preemption*.

Preemption: A preemptive priority scheme is where a customer of a higher class interrupts and ejects a lower-classed customer from service. Three types of preemptive schemes are available: *preemptive–resume, preemptive repeat–identical* and *preemptive repeat–different*. The preemptive–resume scheme allows preempted customers to continue service from their initial point of interruption. The other two named schemes require customers to start again after preemption. The repeat–identical scheme allows the customer to begin again with the full amount of service time required, whereas the repeat–different scheme gives a random service time which does not take into account the time lost. Customers of the same class are served on a FCFS basis.

Non-preemption: The non-preemptive scheme is where a customer of a higher class must wait for a lower-classed customer to complete service if the latter was found to be in service upon the arrival of the higher-class customer.

One of the key objectives of this chapter is to highlight how different service disciplines have been used in communication systems and to compare their performance. We remark that the most mathematically

tractable system is the system $(M/M/c/K)$ with *FCFS* scheduling. Queueing systems with either $A = M$ and $B = GI$ or $A = GI$ and $B = M$ can be analyzed by the method of embedded Markov chains [52.22]. For more general systems, analytical results are often difficult or impossible to obtain. Such systems are conveniently solved using simulation techniques.

52.1.2 Performance Metrics and Traffic Variables

Most queueing models of communication systems assume that the prevailing conditions and constraints on the underlying processes are such that an *equilibrium steady state* is reached and one is dealing with a stationary situation. The analyses can then dispense with the time factor t. This situation will occur if the underlying process is *ergodic*, i. e. aperiodic, recurrent and non-null [52.19], and the system has been in operation for a long time.

The following is a glossary of some performance and traffic variables associated with the simple queue:

- λ : mean arrival rate of customers to the queue. This is usually assumed to be a constant, although it is generally a function of time t when the arrival process is nonstationary.
- μ : mean service rate of customers. This has the same qualification as for λ.
- ρ : traffic intensity. Defined by $\rho = \frac{\lambda}{p\mu}$ for a single class of customer arriving to a queue manned by p servers.
- L_q : the number of customers in the queue in the equilibrium state.
- L_s : the number of customers in the system in the equilibrium state, i. e. which also includes the customers being served.
- W_q : waiting time of each customer in the queue in the equilibrium state.
- W_s : waiting time of each customer in the system in the equilibrium state. In communication systems, W_s is also referred to as the *flow time*. Note that $E(W_s) = E(W_q) + E(X)$, where X is the service time of each customer and $E(\cdot)$ is the expected value operator.
- S : slowdown, defined by $S = W_q/X$. The expected slowdown, $E(S) = E(W_q)E(X^{-1})$ is an important metric in communication systems and measures the expected waiting time of each packet relative to its service requirement. This provides a fairer assessment of the delay suffered by a packet in a queueing system than the waiting time.

The following very useful results, known as Little's formulae, were established as a *folk theorem* for many years until they were shown to be valid by *Little* [52.23]:

$$E(L_q) = \lambda E(W_q)\,,$$
$$E(L_s) = \lambda E(W_s)\,. \qquad (52.1)$$

For queues with finite capacity, λ in (52.1) is replaced by the *effective arrival rate* $\lambda_{\text{eff}} = \lambda \times [1 - \Pr(\text{queue full})]$.

52.1.3 The Poisson Process and the Exponential Distribution

Consider the arrival point process $(T_n)_{n\geq 1}$ and let $N(t)$ be defined as follows:

$$N(t) = \#\{n : T_n \in [0, t)\}$$

i. e. the number of arrivals that occur in the time interval $[0, t)$.

Definition 52.1

The point process $N(t)$ is a stationary Poisson process with rate $\lambda > 0$ if it has the following properties:

a) for any sequence of time points $0 = t_0 < t_1 < t_2 \ldots < t_n$, the process increments $N(t_1) - N(t_0)$, $N(t_2) - N(t_1)$, $\ldots$, $N(t_n) - N(t_0)$ are independent random variables;
b) for $s \geq 0$ and $t \geq 0$, the random variable $N(t+s) - N(t)$ has the Poisson distribution

$$P(N(t+s) - N(t) = j) = \frac{(\lambda s)^j \, e^{-\lambda s}}{j!}\,.$$

There is a close relationship between the exponential random variable and the Poisson process. Suppose the renewal sequence $(\tau_n)_{n\geq 1}$ has an exponential distribution with rate λ, i. e.

$$f(t) = \lambda e^{-\lambda t}\,,\ t \geq 0\,.$$

It can easily be shown that T_k has an Erlang distribution with k stages, i. e.

$$f_k(t) = \frac{\lambda(\lambda t)^{k-1} e^{-\lambda t}}{(k-1)!}\,,\ t \geq 0\,.$$

Therefore,

$$\Pr(T_k \leq t) = 1 - \sum_{j=0}^{k-1} \frac{e^{-\lambda t}(\lambda t)^j}{j!} \qquad (52.2)$$

and, on using the obvious identity

$$\Pr(T_k \le t) = \Pr(N(t) \ge k) \tag{52.3}$$

we obtain

$$\begin{aligned}\Pr(N(t)=k) &= \Pr[N(t)\ge k] - \Pr[N(t)\ge k+1] \\ &= \Pr(T_k \le k) - \Pr(T_{k+1} \le t) \\ &= \frac{(\lambda t)^k \mathrm{e}^{-\lambda t}}{k!}\end{aligned}$$

after applying (52.2). Therefore a renewal sequence with an exponential distribution is a Poisson process. The converse also holds by applying (52.3) and assuming $N(t)$ is a Poisson process.

The Lack of Memory Property of the Exponential Random Variable

Amongst continuous random variables, the exponential random variable has the distinction of possessing the *lack of memory property* (for discrete random variables, the geometric random variable has that property). This property makes analysis of Markovian queues tractable.

The lack of memory property for X states that for any $t > 0$ and $s > 0$,

$$\Pr(X > t+s | X > t) = P(X > s)\,. \tag{52.4}$$

The exponential random variable clearly satisfies (52.4). Therefore, if the time X between consecutive occurrences of a Poisson process has already exceeded t, the chance that it will exceed $s+t$ does not depend on t. Thus the Poisson process *forgets* how long it has been waiting. From (52.4), we also have the following identity

$$\begin{aligned}\Pr(T > t+\Delta t | T > t) &= \mathrm{e}^{-\lambda \Delta t} \\ &= 1 - \lambda\Delta t + o(\Delta t)\,, \end{aligned} \tag{52.5}$$

which implies

$$\begin{aligned}\Pr(T \le t+\Delta t | T > t) &= 1 - \mathrm{e}^{-\lambda\Delta t} \\ &= \lambda\Delta t + o(\Delta t)\,, \end{aligned} \tag{52.6}$$

where $\Delta t \ll t$. Note that (52.5) is the probability of no events within the time interval $(t\,, t+\Delta t)$ given that the time from the last event to the next exceeded t, and (52.6) is the probability of an event within the time interval $(t\,, t+\Delta t)$ given that the time from the last event to the next exceeded t. Therefore, the probability of at least two events within $(t\,, t+\Delta t)$ is

$$1 - [1 - \lambda\Delta t + o(\Delta t)] - [\lambda\Delta t + o(\Delta t)] = o(\Delta t)\,, \tag{52.7}$$

i. e. in a small interval $(t\,, t+\Delta t)$, the Poisson process can increase by at most one occurrence with a non-negligible probability.

52.1.4 Continuous-Time Markov Chain (CTMC)

The theory of Markov process is fundamental in queueing theory and in other branches of applied probability. A comprehensive and authoritative account of the theory can be found in [52.24]. For our purpose, only some rudimentary knowledge of the theory is required and the results here will be presented without proofs.

Definition 52.2

A stochastic process X_t, $t \ge 0$, taking values in a discrete set S, which we may take as the set of non-negative integers, is a standard CTMC if, for any $n = 0, 1, \ldots$ and $t_0 < t_1 < t_2 < \ldots < t_n < t$ and values $i_0, i_1, \ldots, i_n$ and $j \in S$, the following identity holds:

$$\begin{aligned}&\Pr(X_t = j | X_{t_n} = i_n\,, X_{t_{n-1}} = i_{n-1}\,, \ldots\,, X_{t_1} \\ &\quad = i_1\,, X_{t_0} = i_0) = \Pr(X_t = j | X_{t_n} = i_n)\,. \end{aligned} \tag{52.8}$$

A CTMC is said to be *stationary* if, for all $(i, j) \in S \times S$, the one-step transition probability $\Pr(X_{t+h} = j | X_h = i)$ is independent of h and we denote this probability by $p_{ij}(t)$. One-step transition probabilities for stationary CTMC satisfy:

1. $p_{ij}(t) \ge 0$ for all $(i, j) \in S \times S$ and $t \ge 0$.
2. $\sum_{j \in S} p_{ij}(t) = 1$ for all $i \in S$ and $t \ge 0$.
3. $$\lim_{t \to 0+} p_{ij}(t) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \ne j \end{cases}\,.$$
4. (Chapman–Kolmogorov equation)
$$p_{ij}(t+s) = \sum_{k \in S} p_{ik}(t) p_{kj}(s)\,.$$

Theorem 52.1

Let X_t, $t \ge 0$, be a stationary CTMC with transition probability functions $p_{ij}(t)$, $(i, j) \in S \times S$. Then the following (right) derivatives at $t = 0$ exist:

$$\lim_{t \to 0+} \frac{[p_{ii}(t) - 1]}{t} = -q_i$$

$$\text{and} \lim_{t \to 0+} \frac{p_{ij}(t)}{t} = q_{ij} (i \ne j)\,.$$

The parameters q_{ij} and q_i are the *infinitesimal rates* of the CTMC. We note that $0 \le q_i \le \infty$. State i is an *absorbing* state if $q_i = 0$, and it is an *instantaneous* state if $q_i = \infty$. However, q_{ij}, $j \ne i$ is always finite.

Definition 52.3
The matrix $\boldsymbol{A} = (a_{ij})$ where

$$a_{ij} = \begin{cases} -q_i & \text{if } i = j \\ q_{ij} & \text{if } i \ne j \end{cases} \tag{52.9}$$

is called the *infinitesimal generator matrix* $\mathbf{A}$ of the CTMC X_t.

Definition 52.4
A CTMC process is *conservative* if

$$\sum_{j \ne i} q_{ij} = q_i < \infty \text{ for all } i \in S\,. \tag{52.10}$$

The sample path of a conservative CTMC can be described fairly succinctly. If the process is in state i, it remains there for a random time T_i which is exponentially distributed with cumulative distribution

$$\Pr(T_i \le t) = 1 - \mathrm{e}^{-q_i t}\,. \tag{52.11}$$

This is because, by the Markov property (52.8), the probability that the process next jumps to another state depends only on the last recorded state and not on how long it has been in that state. Hence, this lack of memory implies that T_i is an exponential random variable. Furthermore, given that the process is in state i, it next jumps to state $j \ne i$ with probability

$$P_{ij} = \frac{q_{ij}}{q_i}\,. \tag{52.12}$$

Therefore, by considering all situations that could occur, it follows that when $i \ne j$

$$p_{ij}(t) = \sum_{k \ne i} \frac{q_{ik}}{q_i} \int_0^t q_i\, \mathrm{e}^{-q_i s}\, p_{kj}(t-s)\,\mathrm{d}s \tag{52.13}$$

and when $i = j$

$$p_{ii}(t) = \mathrm{e}^{-q_i t} + \sum_{k \ne i} \frac{q_{ik}}{q_i} \int_0^t q_i\, \mathrm{e}^{-q_i s}\, p_{kj}(t-s)\,\mathrm{d}s\,. \tag{52.14}$$

Theorem 52.2
A conservative CTMC satisfies the following system of differential equations called the *forward equations*:

$$p'_{ij}(t) = \sum_{k \ne j} p_{ik}(t) q_{kj} - p_{ij}(t) q_j,\ (i, j) \in S \times S\,. \tag{52.15}$$

If we define the matrix $\boldsymbol{P}(t) = [p_{ij}(t)]$, (52.15) may be concisely expressed as

$$\boldsymbol{P}'(t) = \boldsymbol{P}(t)\boldsymbol{A}\,. \tag{52.16}$$

Let the state distribution of a CTMC X_t be denoted by $\pi(t) = [\pi_i(t)]$, where $\pi_i(t) = \Pr[X(t) = i]$, $i \in S$. Since

$$\pi(t) = \pi(0)\boldsymbol{P}(t)$$

we obtain from (52.16) the equation

$$\pi'(t) = \pi(t)\boldsymbol{A} \tag{52.17}$$

by pre-multiplying both sides of (52.16) with $\pi(0)$.

If the CTMC is ergodic, i. e. aperiodic, recurrent and non-null [52.19], the limit

$$\lim_{t \to \infty} \pi(t) = \bar{\pi}$$

exists and the rate of convergence is exponentially fast [52.24]; $\bar{\pi}$ is known as the *steady-state* or *equilibrium* distribution of the CTMC. Using (52.17), the steady-state distribution is obtained by solving the following system of equations called the *balance equations*:

$$\bar{\pi}\boldsymbol{A} = \boldsymbol{0}\,. \tag{52.18}$$

Birth–Death Processes

A *birth–death process* is a stationary conservative CTMC with state space S, the set of non-negative integers, for which the infinitesimal rates are given by

$$\begin{aligned} q_{n,n+1} &= \lambda_n\,, \\ q_{n,n-1} &= \mu_n\,, \\ q_n &= \lambda_n + \mu_n\,, \\ \text{and } q_{ij} &= 0\,,\ |i-j| \ge 2\,. \end{aligned}$$

A birth and death process can only move to adjacent states, with rates depending only on the state it has moved from (refer to Fig. 52.5). This CTMC is used to model queues where the renewal sequences $(\tau_n)_{n \ge 1}$ and $(s_n)_{n \ge 1}$ are exponential random variables. More

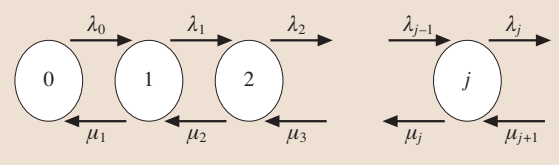

Fig. 52.5 Rate diagram of a birth–death process

specifically, the queueing processes $N_s(t)$ and $N_q(t)$ are birth–death CTMC since they can increase or decrease by 1 with an arrival to or departure from the queueing system, respectively.

From (52.17), the forward equations of the birth–death process take the form

$$\begin{aligned}\pi_0'(t) &= \mu_1\pi_1(t) - \lambda_0\pi_0(t)\,,\\ \pi_j'(t) &= \lambda_{j-1}\pi_{j-1}(t) - (\mu_j+\lambda_j)\pi_j(t)\\ &\quad + \mu_{j+1}\pi_{j+1}(t)\,,\ j\geq 1\,.\end{aligned} \tag{52.19}$$

The solution of (52.19) can be obtained through some ingenious arguments (see e.g. Chap. 4 in [52.13]) but it is much simpler to solve the corresponding balance equations (52.18) for the steady-state distribution $\bar{\pi} = (\pi_0\,,\pi_1\,,\ldots\,,\pi_j\,,\ldots)$, which are:

$$\begin{aligned}\lambda_0\pi_0 &= \mu_1\pi_1\,,\\ (\mu_j+\lambda_j)\pi_j &= \lambda_{j-1}\pi_{j-1} + \mu_{j+1}\pi_{j+1}\,,\ j\geq 1\,.\end{aligned} \tag{52.20}$$

If

$$\sum_{j=0}^{\infty}\frac{\lambda_j\lambda_{j-1}\ldots\lambda_0}{\mu_j\mu_{j-1}\ldots\mu_1} < \infty$$

then (52.20) admits the solution

$$\pi_j = \frac{\lambda_j\lambda_{j-1}\ldots\lambda_0}{\mu_j\mu_{j-1}\ldots\mu_1}\pi_0\,, \tag{52.21}$$

where

$$\pi_0 = \left(1+\sum_{j=0}^{\infty}\frac{\lambda_j\lambda_{j-1}\ldots\lambda_0}{\mu_j\mu_{j-1}\ldots\mu_1}\right)^{-1}\,. \tag{52.22}$$

In the next section, we apply the birth–death process to a very simple queueing model. This is done mainly to illustrate the calculations using the equations displayed in this section as well as to prepare the reader for an analogous but more involved analysis when we come to discuss the MPDQ.

Finite-Buffer Markov Queue (*M*/*M*/*c*/*K*)

This is a very old queueing model with application to telephony and is also commonly used to model other communication systems. Here, it is assumed that the facility can accommodate K customers, including the ones in service, and that once the service facility is full, new arrivals are not allowed in, i. e. they are *lost*. Arrivals to the system are generated according to a Poisson process with rate λ and service time is exponentially distributed with rate μ. There are c servers (or *lines*) where $c \leq K$. From the description of the model, it follows that

$$\lambda_j = \begin{cases}\lambda & \text{for } 0\leq j < K\\ 0 & \text{for } j\geq K\end{cases} \tag{52.23}$$

and

$$\mu_j = \begin{cases}j\mu & \text{for } 0\leq j < c\\ c\mu & \text{for } c\leq j\leq K\,.\end{cases} \tag{52.24}$$

Therefore, (52.21) and (52.22) give

$$\pi_j = \begin{cases}\pi_0\frac{(\rho c)^j}{j!} & \text{for } 0\leq j < c\\ \pi_0\frac{\rho^j c^c}{c!} & \text{for } c\leq j\leq K\,,\end{cases} \tag{52.25}$$

where $\rho = \frac{\lambda}{c\mu}$. Furthermore, applying (52.22), we obtain

$$\pi_0 = \begin{cases}\left(\sum_{j=0}^{c-1}\frac{(c\rho)^j}{j!} + \frac{(c\rho)^c(1-\rho^{K-c+1})}{c!(1-\rho)}\right)^{-1} & \text{if } \rho\neq c\\ \left(\sum_{j=0}^{c-1}\frac{(c\rho)^j}{j!} + \frac{(c\rho)^c(K-c+1)}{c!}\right)^{-1} & \text{if } \rho = c\,.\end{cases} \tag{52.26}$$

Note that π_K is the probability that a customer will be lost (or a packet *dropped*). The steady-state distribution can be used to obtain performance metrics such as:

$$E(L_s) = \sum_{j=1}^{K} j\pi_j$$

$$\text{and } E(L_q) = \sum_{j=c+1}^{K}(c-j)\pi_j$$

whence $E(W_s)$ and $E(W_q)$ are easily derived using the Little formulae (52.1) with λ_{eff} replacing λ.

52.2 Multiple-Priority Dual Queue (MPDQ)

The dual queue with finite buffer is illustrated in Fig. 52.1. We let c_1 be the capacity of the primary queue and c_2 be the capacity of the secondary queue. We assume arrivals belong to k differentiated classes of customers. All arrivals follow an independent Poisson process with rate λ_i and service times are independently exponentially distributed with rate μ_i for $i = 1, 2, \ldots, k$. Unlike the standard dual-queue analysis, we will be incorporating *class-based* service disciplines within the dual-queue model. This means that the favored (high) class of customers will jump to the head of the line before any other class of customers *within the same* queue. Therefore a lower-class customer in the primary queue cannot be ejected to the secondary queue even in the presence of higher-classed customers within the secondary queue. This differs from a single queue in that all high-class customers *in the system* are moved to the head of the queue, whereas in a dual queue, all the high-class customers *within each queue* are moved to the head of the line. A full primary queue blocks any entry from the secondary queue, hence there is an opportunity for lower-class customers to leave the system first, even in the presence of higher-class ones.

In the next section, we compare performances between single (SQ) and dual-queueing models based mainly on waiting-time analysis through simulations using the preemptive and non-preemptive service disciplines (preMPDQ and npMPDQ, respectively). We then give an overview of the analytical solutions when there are only two classes of customers in a subsequent section.

52.2.1 Simulating the MPDQ

The four scheduling disciplines used here in both the single- and dual-queue simulations are FCFS, LIFO, lowest class first (LCF) and highest class first (HCF). LCF and HCF are the two priority disciplines used in the MPDQ. In HCF (and the description is similar for LCF as the two disciplines are symmetrical), a higher-class customer jumps to the head of the queue, behind any already present customers of the same or higher class, and, in turn, is in front of any lower-class customer. As we will see from the simulation results, having both queues under the HCF regime in a dual queue is an effective form of traffic congestion control. The partitioning of a single queue and restricting entry into the primary queue, when it is full, allows lower-class customers to leave the system, even in the presence of other higher-class customers in the system. This is perceived as *fairness* to lower-class customers, as they can exit the system in the presence of higher-class customers but could be deemed as *unfair* to the latter. However, this can only occur when the primary queue is full, so higher-class customers are not disadvantaged all the time.

All simulations on the MPDQ were conducted using the Arena simulation software package [52.25], a flexible windows-based program suitable for a vast array of statistical analysis with dedicated queueing subroutines. In constructing the Arena model, two algorithms (for the preemptive and non-preemptive cases) were designed for the MPDQ. The single-queue schemes were also designed so that various regimes could be compared. Although our simulations dealt only with two classes of customers, which we label class 1 and class 2, all algorithms are simple to extend to multiple classes ($k \geq 3$) of customers. In communication systems, class 1 would refer to high-priority traffics such as web and voice traffic, and class 2 would refer to low-priority traffics such as file transfer protocol (FTP) and hypertext transfer protocol (HTTP).

Results of Simulation Experiments

All customers are considered to be of uniform length. For all models the first-class customer is considered the most infrequent arrival and longest in service. The ra-

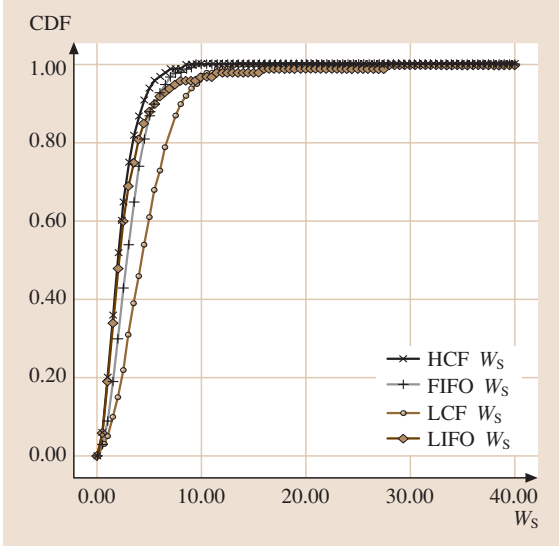

Fig. 52.6 Cumulative distribution functions (CDF) of waiting times for different scheduling disciplines

tionale is that first-class traffic will in many cases be the more demanding on system resources and can be seen as either the most or least valuable, depending upon the type of queueing discipline. Examples of high-class high-demand traffic include video-conference links, streaming audio or streaming video. As more classes are introduced, the performance differences between them may not be as apparent as when there are fewer classes.

The first analysis was to determine whether the HCF regime was superior to the LIFO, FCFS and LCF scheduling disciplines. Our first series of simulations involved the preemptive MPDQ model, the simplest of the two service regimes to simulate. We found that, as traffic intensity increased, the waiting time in the system was markedly lower under a HCF regime for class 1 customers. An example of this is seen in Fig. 52.6, for the parameters $\lambda_1 = 1, \lambda_2 = 2, \mu_1 = 1, \mu_2 = 2, c_1 = c_2 = 4$, the HCF has a lower probability of waiting in the system for class 1 customers in comparison to the other disciplines.

Loss probabilities were also examined. We found marginal differences between the regimes with respect to each class, and little difference in class-based loss. With no appreciable increase in loss by prioritizing traffic, and significant advantages in waiting time, the HCF discipline was chosen as the superior scheduling regime. Further simulations with non-preemptive models continue to confirm that HCF outperformed other scheduling regimes.

The next objective was to explore the differences between the non-preemptive and preemptive service disciplines. The results here provide a framework for communication providers in determining how service disciplines contribute to loss and delay, and how modeling, through simulation, can assist in reducing traffic congestion. We expect class 2 customers to be disadvantaged (in terms of expected waiting time) under the preemptive regime over class 1 (frequent ejection means longer waits) with the reverse situation under the non-preemptive regime. The results certainly show this to be true and Fig. 52.7 clearly demonstrates this.

What improvement does the npMPDQ offer over the non-preemptive single queue (SQ)? We compare the two by considering the non-preemptive dual queue with $c_1 = c_2 = 10$ and the SQ with capacity $c_1 + c_2 = 20$ with a view to measuring its performance under different traffic intensities, achieved through varying the mean arrival rate of class 2 customers. The graphs in Fig. 52.8 are based on a fixed class 1 mean arrival rate, and fixed mean service rates, being $\lambda_1 = 0.2$, and $\mu_1 = \mu_2 = 1$ respectively. We have varied the values of λ_2 from 0.25 to 20. This gives us scope to consider the performance of the system when the traffic intensity is low ($\rho = 0.45$) to when it is well beyond saturation.

From Fig. 52.8, we see that, in periods of low traffic intensity ($\lambda_2 < 1$), there is very little difference in the ratio of class 1 to class 2 customers in the npMPDQ and SQ models. However, for high traffic in-

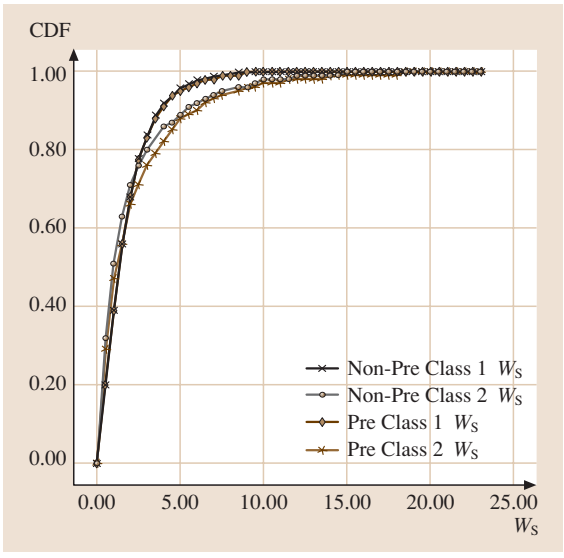

Fig. 52.7 Cumulative distribution functions (CDF) of waiting times for preemptive and non-preemptive service disciplines

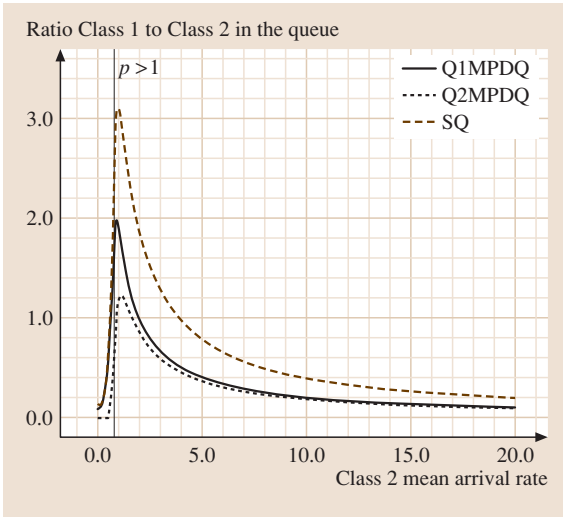

Fig. 52.8 Ratio of class 1 to class 2 customers

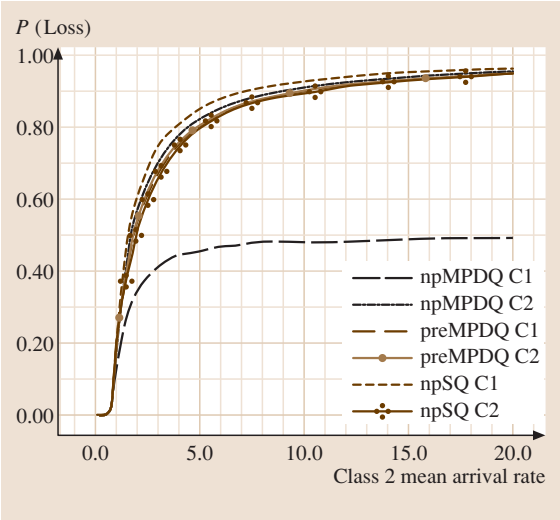

Fig. 52.9 Probability of loss

tensity ($\lambda_2 > 1$) and for both primary and secondary queues, the graphs of the MPDQ models peak and then decline much faster than the SQ model. Note also that, as traffic intensity increases, the distributions of the two classes in both queues appear to stabilize.

Finally we look at the probability of a loss. In Fig. 52.9, we note that class 1 non-preemptive MPDQ customers do best using this measure compared to all the others. The loss probability tapers off significantly for class 1 customers compared to both the SQ and preMPDQ. This does come at the expense of class 2 customers. However, the loss probability for class 2 is still superior to that of the class 1 SQ.

52.2.2 Solving the MPDQ Analytically

In this section, we give a brief summary of the work that has been undertaken in solving the MPDQ for its stationary distribution, and the derivation of the expected waiting times for both classes of customers ([52.12] and [52.26] respectively). We will assume the HCF discipline with class 1 customers designated as the high class. Furthermore, we will only consider the preMPDQ case here, the corresponding results for the npMPDQ case can be obtained from [52.27].

The State Space of the MPDQ and its Infinitesimal Generator

In order to solve the balance equations, which describe the movements by customers between and within the dual queues, it is necessary to define the state space S of the system and the infinitesimal generator matrix A containing the transition rates between states.

State space. The space S can be partitioned into two disjoint sets, $S = S_1 \cup S_2$, corresponding to the case when the secondary queue is empty and when it is not empty, respectively. Here,

$$S_1 = \{(i, j) : 0 \le i + j \le c_1\} ,$$

where i is the number of customers of class 1, and j is the number of customers of class 2. Similarly,

$$S_2 = \{(i, i', j') : 0 \le i' + j' \le c_2, i = 0, 1, \ldots , c_1\} ,$$

where i' is the number of customers of class 1 and j' is the number customers of class 2 in the secondary queue. Note that the number of class 2 customers in the primary queue is simply $c_1 - i$ in this case.

The states of the system can be labeled using S_1 and S_2 above according to the following lexicographical scheme:

$$\boldsymbol{i} = \{(i, 0) , (i, 1) , \ldots , (i, c_1 - i)\}$$
$$\boldsymbol{i0} = \{(i, 0, 1) (i, 0, 2) , \ldots , (i, 0, c_2)\}$$

and for $j = 1, 2, \ldots , c_2$

$$\boldsymbol{ij} = \{(i, j, 0) , (i, j, 1) , \ldots , (i, j, c_2 - j)\}$$

$i = 0, 1, 2, \ldots , c_1$.

The steady-state distribution vector $\overline{\pi}$ is thereby constructed so that its components are ordered using the above labeling scheme:

$$\overline{\pi}^t = \left(\overline{\pi}^t_{\mathbf{0}}, \overline{\pi}^t_{\mathbf{0,0}}, \ldots , \overline{\pi}^t_{\mathbf{0,c_2}}, \overline{\pi}^t_{\mathbf{1}}, \overline{\pi}^t_{\mathbf{1,0}}, \ldots , \overline{\pi}^t_{\mathbf{1,c_2}}, \ldots , \overline{\pi}^t_{\mathbf{c_1}}, \overline{\pi}^t_{\mathbf{c_1,0}}, \ldots , \overline{\pi}^t_{\mathbf{c_1,c_2}}\right) .$$

Note that the dimension of $\overline{\pi}$ equals $\frac{1}{2}(c_1 + 1)(c_2^2 + 3c_2 + c_1 + 2)$. The components of the above steady-state distribution can be described as follows: $\overline{\pi}_{\boldsymbol{i}}$ is the probability vector that is defined when there are no customers present in the secondary queue, whereas $\overline{\pi}_{\boldsymbol{ij}}$ is the probability vector that is defined when there are customers present in the secondary queue (so the primary queue is full). Any probability that is a component of the first type of vector has general form $\pi_{i,j}$ and any probability that is a component of the second type of vector has general form $\pi_{i,i',j'}$. For every tuple listed above, there is an additional label s representing the class of customer *in service* in case the MPDQ is non-preemptive. Thus, the dimension of the steady-state distribution vector is increased to $(c_1 + 1)(c_2^2 + 3c_2 + c_1 + 2) + 1$.

The infinitesimal generator matrix ***A.*** Since the Markov process describing the MPDQ is an ergodic CTMC, the steady-state distribution $\overline{\pi}^t$ exists and is obtained by solving the system of balance equations

$$\overline{\pi}^t \boldsymbol{A} = \boldsymbol{0}\,, \tag{52.27}$$

where $\boldsymbol{0}$ is the zero vector and $\boldsymbol{A}$ is the infinitesimal generator matrix of the process. From the description of the dual-queueing system, $\boldsymbol{A}$ can be partitioned into submatrices whose detailed structure will be described in detail below. The general structure of $\boldsymbol{A}$ is given by

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{\Lambda}_0 & \boldsymbol{\Pi}_0 & 0 & \dots & \dots & 0 \\ \boldsymbol{\Omega}_1 & \boldsymbol{\Lambda}_1 & \boldsymbol{\Pi}_1 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \boldsymbol{\Omega}_{c_1-1} & \boldsymbol{\Lambda}_{c_1-1} & \boldsymbol{\Pi}_{c_1-1} \\ 0 & \dots & & 0 & \boldsymbol{\Omega}_{c_1} & \boldsymbol{\Lambda}_{c_1} \end{pmatrix},$$

where

$$\boldsymbol{\Lambda}_0 = \begin{pmatrix} \boldsymbol{D}_0 & \boldsymbol{M}_{0,\lambda_2} & \boldsymbol{M}_{0,\lambda_1} & 0 & & 0 \\ \boldsymbol{O}_{0,\mu_2} & \boldsymbol{D}_{0,0} & \boldsymbol{Q}_{0,\lambda_1} & 0 & & \\ 0 & 0 & \boldsymbol{D}_{0,1} & \boldsymbol{R}_{0,1,\lambda_1} & & \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ & & & \ddots & \ddots & \boldsymbol{R}_{0,c_2-1,\lambda_1} \\ 0 & & \cdots & & 0 & \boldsymbol{D}_{0,c_2} \end{pmatrix},$$

$$\boldsymbol{\Pi}_0 = \begin{pmatrix} \boldsymbol{N}_{0,\lambda_1} & 0 & & \cdots & & 0 \\ 0 & 0 & & & & \\ \boldsymbol{S}_{0,\mu_2} & \boldsymbol{U}_{0,1} & \ddots & & & \vdots \\ 0 & 0 & \ddots & \ddots & & \\ \vdots & & \ddots & \ddots & \ddots & \\ 0 & \cdots & & 0 & \boldsymbol{U}_{0,c_2} & 0 \end{pmatrix},$$

and for $i = 1, 2, \dots, c_1$

$$\boldsymbol{\Lambda}_i = \begin{pmatrix} \boldsymbol{D}_i & \boldsymbol{M}_{i,\lambda_2} & \boldsymbol{M}_{i,\lambda_1} & 0 & & \cdots & 0 \\ 0 & \boldsymbol{D}_{i,0} & \boldsymbol{Q}_{i,\lambda_1} & 0 & & & \\ \boldsymbol{S}_{i,\mu_1} & \boldsymbol{U}_{i,1} & \boldsymbol{D}_{i,1} & \boldsymbol{R}_{i,1,\lambda_1} & \ddots & & \vdots \\ 0 & 0 & \boldsymbol{U}_{i,2} & \boldsymbol{D}_{i,1} & \ddots & \ddots & \\ \vdots & & \ddots & \boldsymbol{U}_{i,2} & \ddots & \ddots & 0 \\ & & & \ddots & \ddots & \ddots & \boldsymbol{R}_{i,c_2-1,\lambda_1} \\ 0 & & \cdots & & 0 & \boldsymbol{U}_{i,c_2} & \boldsymbol{D}_{i,c_2} \end{pmatrix},$$

$$\boldsymbol{\Omega}_i = \begin{pmatrix} \boldsymbol{P}_{i,\mu_1} & 0 & \cdots & 0 \\ \boldsymbol{Y}_{i,\mu_1} & \boldsymbol{T}_{i,\mu_1} & & \\ 0 & 0 & & \vdots \\ \vdots & & \ddots & \vdots \\ & & & \\ 0 & & \cdots & 0 \end{pmatrix},$$

and

$$\boldsymbol{\Pi}_i = \begin{pmatrix} \boldsymbol{N}_{i,\lambda_1} & 0 & \cdots & 0 \\ 0 & 0 & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & 0 \end{pmatrix}.$$

Each submatrix represents all state transitions generated by the arrival/departure patterns of the MPDQ and incorporates the rate parameters λ_i and μ_i, $i = 1, 2$. We remark that not every submatrix is a square matrix and hence invertible. For example,

$$D_{i,j} = \begin{pmatrix} -(\mu_1+\lambda) & \lambda_2 & 0 & & \cdots & 0 \\ 0 & -(\mu_1+\lambda) & \ddots & & & \\ & 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \lambda_2 & 0 \\ & & & \ddots & -(\mu_1+\lambda) & \lambda_2 \\ 0 & & & \cdots & 0 & -\mu_1 \end{pmatrix}$$

is a square matrix of dimension $(c_2 - j + 1)$ representing transitions between states in $\boldsymbol{ij}$, $i = 1, 2, \dots, c_1$, $j = 1, 2, \dots, c_2$ while

$$N_{i,\lambda_1} = \begin{pmatrix} \lambda_1 & 0 & \ddots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \vdots & & 0 & \lambda_1 \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$$

is a matrix of dimension $(c_1 - i + 1) \times (c_1 - i)$ representing the transition from (i, j) to $(i+1, j)$ where $i = 0, 1, \dots, c_1 - 1$, $j = 0, 1, \dots, c_1 - i - 1$. Detailed of the other submatrices can be obtained from [52.12].

Technically speaking, any linear numerical procedure could be used to solve (52.27). However, this would ignore the structure of the system and the fine detail of the submatrices outlined above, especially the fact that

not all the submatrices are square matrices and that A is not in a workable block-tridiagonal form. This could render standard numerical procedures difficult or even impossible to apply; so, in [52.12], an algorithm which takes into account the structure of the system is proposed and shown to be very fast and easy to implement.

52.2.3 The Waiting-Time Distribution

The derivation of the waiting-time distribution, especially for class 2 customers, is based on the matrix analytic method pioneered by *Neuts* [52.28]. Our method generalizes the method proposed in [52.29] for the non-preemptive SQ. Matrix analytic methods exploit the special structures of the transition matrices or infinitesimal generators of some Markov processes occurring in queueing models and an important feature of these methods, which add to their utility, is that they are computational in character. In the subsequent analysis, we distinguish between two cases: when the primary queue is not full and when it is.

The Primary Queue is not Full

Class 1 customers: in this case, a tagged class 1 customer C_1 who joins the queue is concerned only with the number of class 1 customers ahead of him. If C_1 sees no class 1 customers ahead of him, then he goes to the head of the line and ejects the class 2 customer, if any, who is being served. If C_1 sees n, $0 < n < c_1$ class 1 customers ahead of him, then he has to wait until these customers have completed their services. Therefore, his waiting time is the sum of n exponential (μ_1) random variables. Let

$$\Pi_{Q_1} = \sum_{0 \le i+j < c_1} \pi_{i,j}$$

i. e. the probability that queue 1 is not full, then the conditional density of the waiting time of C_1 given that queue 1 is not full is

$$W_1^{(1)}(t) = \frac{1}{\Pi_{Q_1}} \left[\left(\sum_{j=0}^{c_1-1} \pi_{0,j} \right) \delta(t) + \sum_{j=0}^{c_1-2} \sum_{i=1}^{c_1-1-j} \pi_{i,j} f_{i,\mu_1}(t) \right] ,$$

where $\delta(t)$ is the Dirac delta function and

$$f_{n,\mu_1}(t) = \frac{\mu_1^n}{(n-1)!} t^{n-1} \mathrm{e}^{-\mu_1 t} ,$$

i. e. the Erlang distribution with n phases and parameter μ_1.

Class 2 customers: the derivation of the waiting time of an arbitrary class 2 customer C_2 entering the secondary queue is quite complex as it is affected by subsequent arrivals of both class 1 and class 2 customers. Suppose C_2 joins the queue and finds at least one customer in front of him, he will be pushed back in the queue by subsequent class 1 arrivals. Also, any subsequent class 2 arrivals will reduce the available spaces in the buffer, thus improving his chances of moving to the front of the line. The waiting time of C_2 will depend on the absorption time into a particular set $\mathcal{A}_1$ [defined by (52.28)] experienced by the stochastic process $Z(t)$, $t \ge 0$, where

$$Z(t) = [l_{11}(t), L_{21}(t), F_1(t)]$$

where $l_{11}(t)$ is the number of class 1 customers in the primary queue, $L_{21}(t)$ is the number of class 2 customers in front and including C_2 in the primary queue, and $F_1(t)$ is the amount of free space in the primary queue. Due to the assumptions underlying the MPDQ, $Z(t)$ is a continuous-time Markov chain taking values in the set

$$\mathcal{R}_1 = \{(l_{11}, L_{21}, F_1) \in \mathcal{N}_0^3 : 0 \le l_{11} \le c_1 , \; 0 \le L_{21} \le c_1 , \; 0 \le F_2 \le c_1 \} .$$

Define the set of states $\mathcal{A}_1 \subset \mathcal{R}_1$ by

$$\mathcal{A}_1 = \{(0, 1, F_1) : 0 \le F_1 \le c_1 - 1\} . \qquad (52.28)$$

Let $T_1(l_{11}, L_{21}, F_1)$ denote the first-passage time for the process $Z(t)$ into $\mathcal{A}_1$ starting initially in state (l_{11}, L_{21}, F_1) where $L_{21} \ge 1$, i. e.

$$T_1(l_{11}, L_{21}, F_1) = \min\{t \ge 0 : Z(t) \in \mathcal{A}_1 | Z(0) = (l_{11}, L_{21}, F_1)\} .$$

Note that this is precisely the time it will take for C_2 to be served. Denote the Laplace transform of $T_1(l_{11}, L_{21}, F_1)$ by $\hat{W}_{l_{11},L_{21},F_1}(s)$, i. e.

$$\hat{W}_{l_{11},L_{21},F_1}(s) = E(\mathrm{e}^{-sT_1(l_{11},L_{21},F_1)}) , \qquad (52.29)$$

where $\mathrm{Re}(s) > 0$. Using the Poisson arrivals see time averages (PASTA) property [52.30], which posits that a Poisson arrival would observe the steady-state distribution at any random time point, the Laplace transform of the time to absorption of $Z(t)$ given that C_2 can join the primary queue is therefore

$$\hat{W}_1^{(2)}(s) = \frac{1}{\Pi_{Q_1}} \left[\sum_{0 \le i+j < c_1} \pi_{i,j} \hat{W}_{i,j+1,c_1-i-j-1}(s) \right] . \qquad (52.30)$$

Inversion of (52.30) will in principle gives us the distribution of the waiting time of a class 2 customer. However, for most practical purposes, it suffice to compute its k-th moment, $k = 1, 2, \ldots$. A recursive and computationally efficient algorithm using matrix analytic methods is introduced in [52.26], which achieved this purpose.

The Primary Queue is Full

Class 1 customers: here, C_1 joining the secondary queue either sees no class 1 customers or at least one class 1 customer in the primary queue. In the first case, he has to wait for the services of all class 1 customers ahead of him in the secondary queue and that of the class 2 customer at the head of the queue to finish. In the second case, he has to wait until all class 1 customers in front of him in the combined queue have been served. Thus the conditional density of his waiting time given that the primary queue is full is

$$W_2^{(1)}(t) = \frac{1}{\Pi_{Q_2}} \left[\sum_{0 \le i'+j' < c_2} \pi_{0,i',j'} f_{1,\mu_2} \star f_{i',\mu_1}(t) + \sum_{i=1}^{c_1} \sum_{0 \le i'+j' < c_2} \pi_{i,i',j'} f_{i+i',\mu_1}(t) \right] \tag{52.31}$$

where $\star$ refers to the convolution operator and

$$\Pi_{Q_2} = \sum_{i=0}^{c_1} \sum_{0 \le i'+j' < c_2} \pi_{i,i',j'}$$

is the probability that the primary queue is full.

Class 2 customers: similar to the first case, the waiting time of an arbitrary class 2 customer C_2 is equal to the absorption time of a stochastic process into a targeted set. Define

$$Y(t) = [l_{11}(t), l_{12}(t), L_{22}(t), F_2(t)] ,$$

where $l_{11}(t)$ is the number of class 1 customers in the primary queue, $l_{12}(t)$ is the number of class 1 customers in the secondary queue, $L_{22}(t)$ is the number of class 2 customers in front and *including* C_2 in the dual queue, and $F_2(t)$ is the amount of free space in the secondary queue. The time for $Y(t)$ to first enter the targeted set $\mathcal{A}_2$ [defined by (52.32)] is equal to the waiting time of C_2. Note that $Y(t)$ is a continuous-time Markov chain which takes values in the set

$$\begin{aligned} \mathcal{R}_2 = \{(l_{11}, l_{12}, L_{22}, F_2) \in \mathcal{N}_0^4 : & 0 \le l_{11} \le c_1 , \\ & 0 \le l_{12} \le c_2 , \\ & 0 \le L_{22} \le c_1 + c_2 , \\ & 0 \le F_2(t) \le c_2 \} . \end{aligned}$$

The waiting time of C_2 is the first-passage time $T_2(l_{11}, l_{12}, L_{22}, F_2)$ for the process Y to enter into $\mathcal{A}_2 \subset \mathcal{R}_2$ defined by

$$\mathcal{A}_2 = \{(0, l_{12}, 1, F_2) : 0 \le l_{12} < c_2, 0 \le F_2 \le c_2\} \tag{52.32}$$

starting initially in state $(l_{11}, l_{12}, L_{22}, F_2)$, where $L_{22} \ge 1$. Denote the Laplace transform of $T_2(l_{11}, l_{12}, L_{22}, F_2)$ by $\hat{W}_{l_{11},l_{12},L_{22},F_2}(s)$. By the PASTA property, the Laplace transform of the time to absorption of Y into $\mathcal{A}_2$ given that a class 2 customer enters the secondary queue is

$$\hat{W}_2^{(2)}(s) = \frac{1}{\Pi_{Q_2}} \left[\sum_{i=0}^{c_1} \sum_{0 \le i'+j' < c_2} \pi_{i,i',j'} \times \hat{W}_{i,i',j'+c_1-i+1,c_2-i'-j'-1}(s) \right] . \tag{52.33}$$

Again, inverting (52.33) will technically gives us the distribution of the waiting time of a class 2 customer. However, as in previous case, a more viable alternative is to provide a recursive algorithm (which we have done in [52.26]) which will allow us to compute all the k-th, $k = 1, 2, \ldots$ moments of the waiting time for class 2 customers in the secondary queue.

52.3 Distributed Systems and Load Balancing

Many recent studies have shown that distributed computing environments exhibit a wide range of task sizes, often spanning many orders of magnitude. These so-called *heavy-tailed* workloads have been found to exist in a number of computing environments. *Crovella* et al. [52.31, 32] found that a number of file-size distributions measured on the World Wide Web (WWW) exhibit heavy tails, including file requests by users, files transmitted via the network and files stored on servers. Further examples of observed heavy-tailed workload include the size of files stored in Unix file systems [52.33] and the Unix process central processing unit (CPU) requirements measured at UC Berkley [52.34]. More recently, traffic measurements of the 1998 World

Cup [52.35] and the 1998 Winter Olympics [52.36] have exhibited some heavy-tailed characteristics. There are significant questions raised by these findings with regards to task assignment policies, as much of the existing work in the area was formulated under an assumption of an exponentially distributed workload.

Heavy-tailed distributions have very high variance, where 1% of tasks can take 50% of the computing resources. These distributions are characterized by the property

$$\Pr(X > x) \sim x^{-\alpha} ,$$

where $0 \leq \alpha \leq 2$. Any set of tasks that is said to follow a heavy-tailed distribution is described as having the following properties [52.37, 38]:

1. Decreasing failure rate. That is, the longer a task has run, the longer it is expected to continue running.
2. Infinite variance, and if $\alpha \leq 1$, infinite mean.
3. A very small fraction (less than 1%) of the very largest jobs make up a large fraction (half) of the workload. This is commonly referred to as the *heavy-tailed property*. It is this property that makes the load very difficult to balance effectively.

For the purpose of analysis, we assume that the task sizes show some maximum (but large) value. This is a reasonable assumption in many cases, such as a web server, which would have some largest file. A *bounded Pareto* distribution is therefore used, which has a lower and upper limit on the task size distribution. The probability density function for the bounded Pareto $B(k, p, \alpha)$ is:

$$f(x) = \frac{\alpha k^{\alpha}}{1-(k/p)^{\alpha}} x^{-\alpha-1}, k \leq x \leq p , \qquad (52.34)$$

where α represents the task size variation, k is the smallest possible task, and p is the largest possible task. By varying the value of α we can observe distributions that exhibit moderate variability ($\alpha \approx 2$) to high variability ($\alpha \approx 1$). Typical measured values of the α parameter are 0.9–1.3 [52.31, 32, 37], with an empirically measured mean of $\alpha \approx 1.1$. The next table provides some α values associated with the heavy-tailed distributions of some files.

52.3.1 Classical Load-Distribution Policies

The problem of optimal task assignment in a distributed system has been a well-researched area for many years. Most of the so-called *classical* approaches were created under the assumption that service times are exponentially distributed. Many of these policies are still widely used, due to their simplistic nature and ease of implementation.

Random and Round-Robin

Classical task assignment policies such as *random* and *round-robin* [52.39] have traditionally been used in distributed systems, and are still widely used for many applications. Under the random policy, tasks are assigned to each back-end server with equal probability. Using a round-robin policy, tasks are assigned to servers in a cyclical fashion. Both policies equalize the expected number of tasks allocated to each server, and are frequently used as a baseline to compare with other task distribution policies. Tasks are assigned with no consideration of each host's load or the distribution of task sizes. Despite this, random and round-robin are still commonly used in many scheduling environments (most likely due to ease of implementation). It has been shown previously [52.34] that random and round-robin both have similar performance characteristics.

Dynamic Policies

Dynamic policies intelligently assign tasks based on knowledge of the current load at each host. The LLF (least loaded first) approach assigns tasks to the server with the least amount of work remaining, attempting to achieve instantaneous load balance. The work remaining can be approximated by the queue length (shortest queue), or assuming the task's service requirement is known a priori. By keeping the load balanced, the waiting time in the queue can be reduced. It is known that balancing the load minimizes the mean response time [52.40] in the type of distributed system that we consider in this paper. Despite this, a number of caveats exist. Firstly, the best performance is not always obtained by balancing the load, particularly if you are interested in different measures of performance, such as the mean slowdown. Secondly, balancing the load is not always practical, as you are often depending on approximate measures of the load, such as the queue length. Under highly variable workload it is highly probable that the length of a queue can be misleading as an indicator of congestion.

Central Queue

The central-queue policy holds tasks in a queue at the dispatcher until a host is idle. Such a policy has proved to be equivalent to a least-work-remaining policy, showing that equivalent performance can be obtained without any prior knowledge of a task's size [52.37, 38]. Recently, two variations of the central-queue policy have

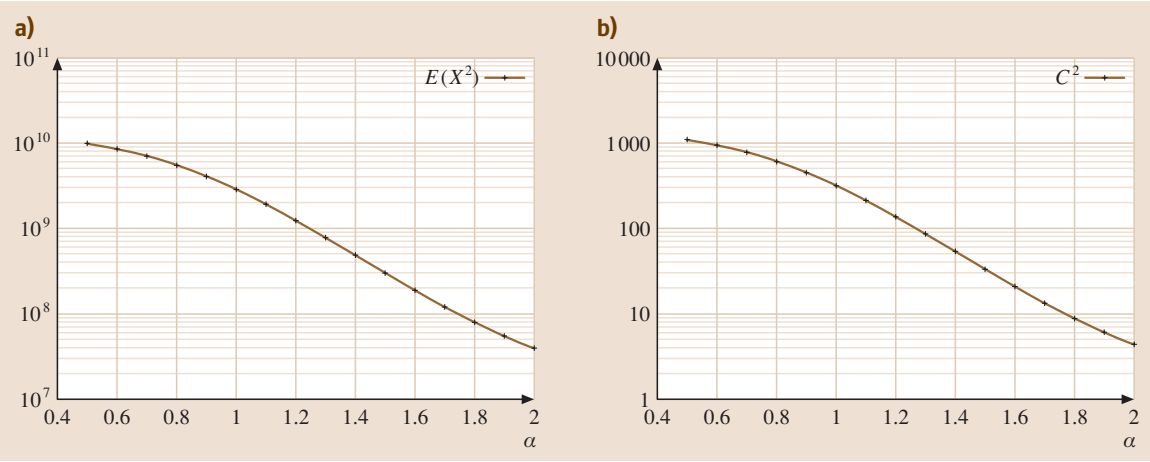

Fig. 52.10a,b The second moment of a bounded Pareto distribution ($E\{X\} = 3000$, $p = 10^7$) is shown in **(a)**, where α is varied from 0.5 to 2.0. The squared coefficient of variation ($C^2 = E\{X^2\}/E\{X\}^2$) is shown in **(b)**

been proposed – cycle stealing with immediate dispatch (CS-ID) and cycle stealing with central queue (CS-CQ) [52.41]. CS-ID immediately dispatches tasks to a back-end server, while CS-CQ holds tasks in a central queue at the dispatcher until a host is idle. Both policies are evaluated against a dedicated policy. In a dedicated policy, one host is *dedicated* to servicing all short jobs, while the other host services long jobs. Both CS-ID and CS-CQ follow a similar arrangement, but can steal cycles from an idle host if available and it is prudent to do so. Both CS-ID and CS-CQ show improvement over a dedicated policy in many areas (notably for short tasks). The application of these policies are limited to domains where a priori knowledge of a tasks size is known, and in the case of CS-CQ, there needs to be constant feedback between the dispatcher and the back-end hosts to notify the dispatcher of an idle host.

Some Known Results

The round-robin policy results in a slightly less variable arrival stream to the back-end hosts than the random policy. Despite this, the performance of the random and round-robin policies have been shown to be roughly equivalent [52.38].

Assuming an $M/M/c$ queueing system, a shortest-queue policy has been shown to be optimal with respect to maximizing the number of jobs completed by a certain time [52.42]. Under more general assumptions, *Nelson* and *Phillips* [52.43] claim that the central-queue (and therefore least-work-remaining) policy is optimal.

A least-work-remaining policy is not analytically tractable under $M/GI/c$ queuing systems. Nonetheless, this policy has been shown to be equivalent to a central-queue policy [52.38], for which there exists known approximations to its queue size using the following result given in [52.44]:

$$E(L_{M/GI/c}) = E(L_{M/M/c})\frac{E(X^2)}{E(X^2)}\,, \tag{52.35}$$

where X is the service requirement, and L_{model} represents the queue length under the model specified. We remark that, for the queueing model (M/GI/1), the expected waiting time in the queue is given by the famous *Pollaczek–Khinchin mean value formula* [52.19]

$$E(W_q) = \frac{\lambda E(X^2)}{2(1-\rho)}\,. \tag{52.36}$$

Limitation of Classical Load Distribution Policies

The task assignment policies listed above are not suitable for heavy-tailed distributions such as the Pareto distribution. Consider the metrics (52.35) and (52.36); what is immediately apparent is that all these metrics depend on the second moment of the service requirement distribution, $E\{X^2\}$ and for policies, such as the least-work-remaining policy, they will depend on the squared coefficient of variation $C^2 = E\{X^2\}/E\{X\}^2$. Figure 52.10 shows an example of one such highly variable distribution (where the y-axis is on a log scale). We can see that, as α decreases, the variation (as represented by the second moment of the distribution, and the squared coefficient of variation) increases substantially. Clearly,

as the variability of the service time distribution increases, the performance of the distributed system will decrease rapidly using these classical load balancing policies.

52.3.2 Size-Based Load Distribution Policies

In the previous section we have highlighted substantial recent research indicating the frequent occurrence of heavy-tailed workloads in many distributed computing environments. The characteristics of these heavy-tailed workloads make them very challenging to manage using traditional load distribution policies. Indeed, many of these policies were created under the $M/M/c$ queueing model, where the distribution of service requirements follows an exponential distribution. To deal with heavy-tailed $M/G/c$ workloads, where G is a heavy-tailed distribution, new load distribution techniques need to be employed. In particular, they must address the characteristics of these workloads, such as their highly variable nature, which cause such poor performance under traditional load distribution policies. In recent years, there have been several load distribution techniques specifically created to exploit the special characteristics of heavy-tailed workloads. They can be broadly classified as *size-based* policies where the workload is partitioned into distinct size ranges, with each size range dedicated to a specific server. For example, you may have a two server system where one server processes only small tasks, while another server processes only large tasks.

These size-based policies can be further classified by what knowledge they assume is known at the dispatcher. Some policies assumes that a task's size is known a priori at the dispatcher, and as such can assign the task directly to the server that is responsible for servicing tasks in that range. This obviously restricts the application of these policies to domains where exact (or reasonably accurate) a priori knowledge of a task's size is available.

Other size-based policies have less restrictive assumptions regarding what information is available at the dispatcher. Policies such as task assignment based on guessing size (TAGS) and task assignment based on prioritizing traffic flows (TAPTF), which are discussed later, assume no knowledge of a task's size at the dispatcher. They do, however, require knowledge of the distribution of task sizes.

SITA-E/V/U – Known Task Size

Size interval task assignment with equal load (SITA-E) [52.38] is a sized-based approach proposed by *Harchol-Balter* et al. that associates a unique size range with each host in the distributed system. These size ranges are chosen specifically to equalize the expected load received at each host. Whilst proving effective under conditions of high task-size variability, SITA-E is not the best policy in circumstances of lower task-size variability, where a dynamic policy is more suitable.

Size interval task assignment with variable load (SITA-V) [52.40] intentionally operates the hosts in a distributed system at different loads, and directs smaller tasks to lighter-loaded servers. The authors note that, depending on which performance metrics are of interest, the conventional notion that balancing the load on the respective hosts may not result in optimal performance, especially when the size distribution is heavy-tailed. SITA-V, like SITA-E, assigns tasks to a given host based on their size. However, SITA-V exploits the heavy-tailed property of the task size distribution by running the vast majority of tasks (i. e. the small tasks) on lightly-loaded hosts, while running the minority of tasks (the larger sized tasks) on the heavily-loaded hosts, thus preventing small tasks getting held up behind large tasks and allowing them to be processed quickly. Thus, mean slowdown is reduced, and the throughput is not adversely affected, but it can result in an increase in mean waiting time – which is expected, since minimal mean waiting time is known to occur when load is balanced.

A size-based approach that is specifically suited for batch computing environments under supercomputing workloads is size interval task assignment with unbalanced load (SITA-U) [52.45]. SITA-U purposely unbalances the load among the hosts while also being *fair*, i. e. achieving the same expected slowdown for all jobs. Two variations of SITA-U are considered: SITA-U-opt, where service requirement cutoffs are chosen to minimize mean slowdown, and SITA-U-fair, where service requirement cutoffs are chosen to maximize fairness. The simulation results showed that both variations of SITA-U performed better under a range of load conditions – with system loads in the range 0.1–0.8. SITA-U-fair achieved significant performance gains over the load range 0.5–0.8, demonstrating an improvement of 4–10 times with regards to mean slow down, and from 10–100 times with regards to variability in slowdown.

Most size-based policies perform well under very high task-size variation, but their advantage over existing approaches is reduced as variation decreases. Most importantly, the application of the task assignment poli-

cies listed above is limited by the assumption that the service requirement of each task is known a priori, which is frequently not the case.

TAGS/TAPTF – Unknown Task Size

The size-based approaches considered thus far all assume that the exact service requirements are known at the dispatcher in advance. Often this is not the case – in many environments a task's service requirement is not known until execution time on a given host. Task assignment based on guessing size (TAGS) [52.37] assumes no prior knowledge of a task's service requirement. Like SITA-V, TAGS is slightly counterintuitive in that it unbalance the load, and also considers the notion of *fairness*, i.e. that all tasks should experience the same expected slowdown. The TAGS approach works by associating a processing time limit with each host. Tasks are executed on a host, starting with host 1 (refer to Fig. 52.2), up until the designated time limit associated with that host; if the task has not completed by this point, it is aborted and restarted from scratch at the next host. These cutoffs are a function of the distribution of task sizes and the external arrival rate, and can be computed to optimize certain metrics, such as waiting time or slowdown.

The design of the TAGS policy purposely exploits properties of the heavy-tailed distribution, such as decreasing failure rate – where the longer a task has run, the longer it is expected to run – and the fact that a tiny fraction (less that 1%) of the very longest tasks can make up over half the load.

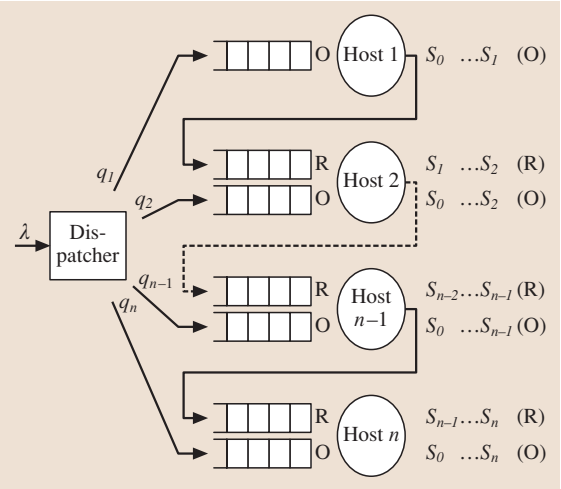

Fig. 52.11 TAPTF queueing system

Like other size-based approaches, under higher loads and less variable conditions, TAGS does not perform so well. TAGS gains much of its performance by exploiting the heavy-tailed property, by moving the majority of the load onto host 2, allowing the vast majority of small tasks to be processed quickly on host 1. TAGS also suffers under high loads due to *excess* – the extra work created by restarting many jobs from scratch. As noted in [52.37], " . . . overall excess increases with load because excess is proportional to λ, which is in turn proportional to load."

Recently a new approach to task assignment in a distributed system, task assignment based on prioritizing traffic flows (TAPTF) [52.46], was introduced (Fig. 52.11). TAPTF is a flexible policy that addresses some of the shortcomings of existing approaches to task assignment. TAPTF showed improved performance under heavy-tailed workloads for certain classes of traffic by controlling the influx of tasks to each host. Tasks can potentially be dispatched to any host, rather than just the first host, as in the TAGS approach. This approach is beneficial under two important scenarios. First, when task size variation decreases it becomes harder to exploit the so-called heavy-tailed properties. When the workload becomes more uniformly distributed, improved performance is gained from spreading the incoming tasks over multiple hosts. Second, when the system load is high the first host (which in a TAGS system receives *all* incoming tasks) can become overloaded, causing a bottleneck and resulting in a decrease in the overall performance metrics.

TAPTF introduces multiple queues with processing time limits (cutoffs). Each host has an ordinary queue which receives tasks directly from the dispatcher. Each host (excluding the first) also has a restart queue that receives restarted tasks from the host directly above it. The use of dual queues (combined with cutoffs at each host) enables service differentiation at each host, allowing smaller tasks to be executed quickly without being delayed by larger tasks. To achieve this, tasks that exceed the cutoff on a given host are migrated to the next host's restart queue (to be restarted from scratch). As the ordinary queue has priority over the restart queue, a potentially small task is not delayed by larger tasks that may exist in the restart queue.

Improved performances were observed both in mean waiting time and mean slowdown, the key areas where TAGS and random policies suffer. Most significantly, TAPTF exhibited improved performance under low to high task size variation and high system load by reducing

the excess associated with a large number of restarts and by intelligently controlling the influx of tasks to each back-end host.

Performance Under Heavy-Tailed Workloads

In this section, we illustrate how such size-based approaches are effective at counteracting the high variability in workloads that can affect the performance of traditional approaches to load balancing. Consider a size-based policy that partitions the workload between the back-end hosts. A size-based policy assigns unique size ranges to each of the n back-end hosts by partitioning the range of task sizes $[k, p]$, i. e. $k = s_0 < s_1 < s_2 < \ldots < s_n = p$. For example, in a two-host system, host 1 would service tasks with sizes between s_0 and s_1, while host 2 would handle the remaining tasks, with sizes between s_1 and $s_2 = p$.

Let p_i equal the fraction of tasks whose destination, i. e. where it will run to completion, is host i. That is, tasks whose size is between s_{i-1} and s_i. Using a bounded Pareto distribution (52.34), this is given by:

$$\begin{aligned} p_i &= P(s_{i-1} \le X \le s_i) \\ &= \frac{\alpha k^\alpha}{1-(k/p)^\alpha} \int_{s_{i-1}}^{s_i} x^{-\alpha-1}\,\mathrm{d}x \\ &= \frac{k^\alpha}{1-(k/p)^\alpha}\left(s_{i-1}^{-\alpha} - s_i^{-\alpha}\right) . \end{aligned} \tag{52.37}$$

Let us now consider only those tasks which are dispatched to and run-to-completion at host i. Let $E\left(X_i^j\right)$ be the j-th moment of the distribution of tasks that are dispatched to host i's queue. We have:

$$E\left(X_{iO}^j\right) = \int_{s_{i-1}}^{s_i} x^j f(x)\,\mathrm{d}x \tag{52.38}$$

$$= \begin{cases} \dfrac{\alpha s_{i-1}^\alpha}{(j-\alpha)\left[1-\left(\frac{s_{i-1}}{s_i}\right)^\alpha\right]} & \text{if } j \neq \alpha \\ \dfrac{s_{i-1}s_i}{s_i - s_{i-1}}(\ln s_i - \ln s_{i-1}) & \text{otherwise .} \end{cases} \tag{52.39}$$

Consider first the following example: a four-host system, utilizing the SITA-E task assignment policy. The arrival is Poisson and the service time distribution follows a bounded Pareto distribution. As described previously, SITA-E chooses its size ranges in order to equalize the expected load assigned to each back-end host. Figure 52.12a shows the C^2 values experienced by each back-end host as α varies. We can see a significant reduction in variation that has been achieved by partitioning the workload and assigning it to different hosts – effectively grouping like-sized tasks together. Indeed, the first two hosts have C^2 values that are less than one from $\alpha = 1.1$ to $\alpha = 2.0$. Nevertheless, we can see that the variation at the latter hosts is still quite high, approaching the value of C^2 of the task size distribution itself (i. e. before it is partitioned). This is not a problem in itself, as we can see in Fig. 52.12b. There, using the formula for p_i given in (52.37), we see that the vast majority of tasks are processed by the lower hosts, and predominantly the first host. These hosts have a significantly lower variance, and, given they process nearly

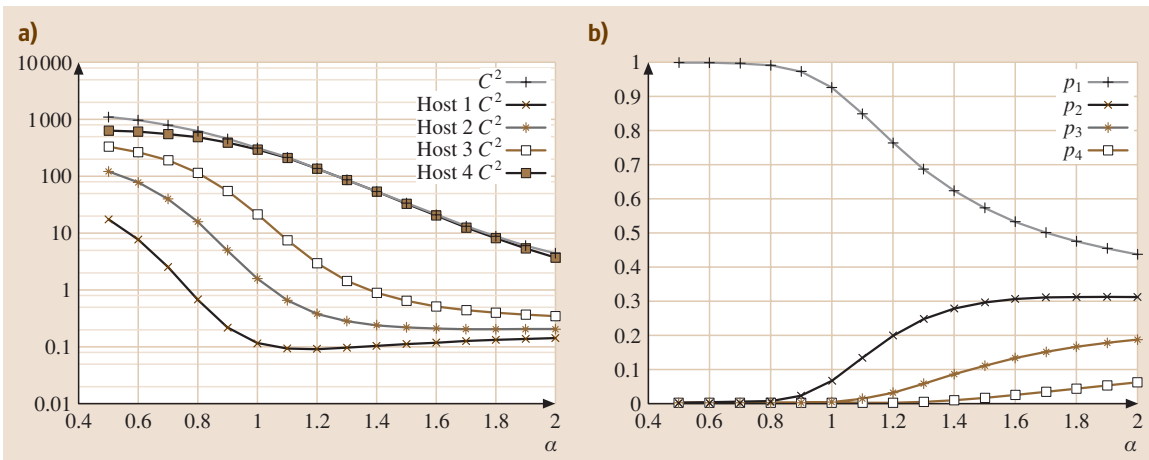

Fig. 52.12a,b The squared coefficient of variation experienced at each host ($C^2 = E\{X^2\}/E\{X\}^2$) in a four-host SITA-E system is shown in **(a)**. The fraction of tasks assigned to each host is shown in **(b)**

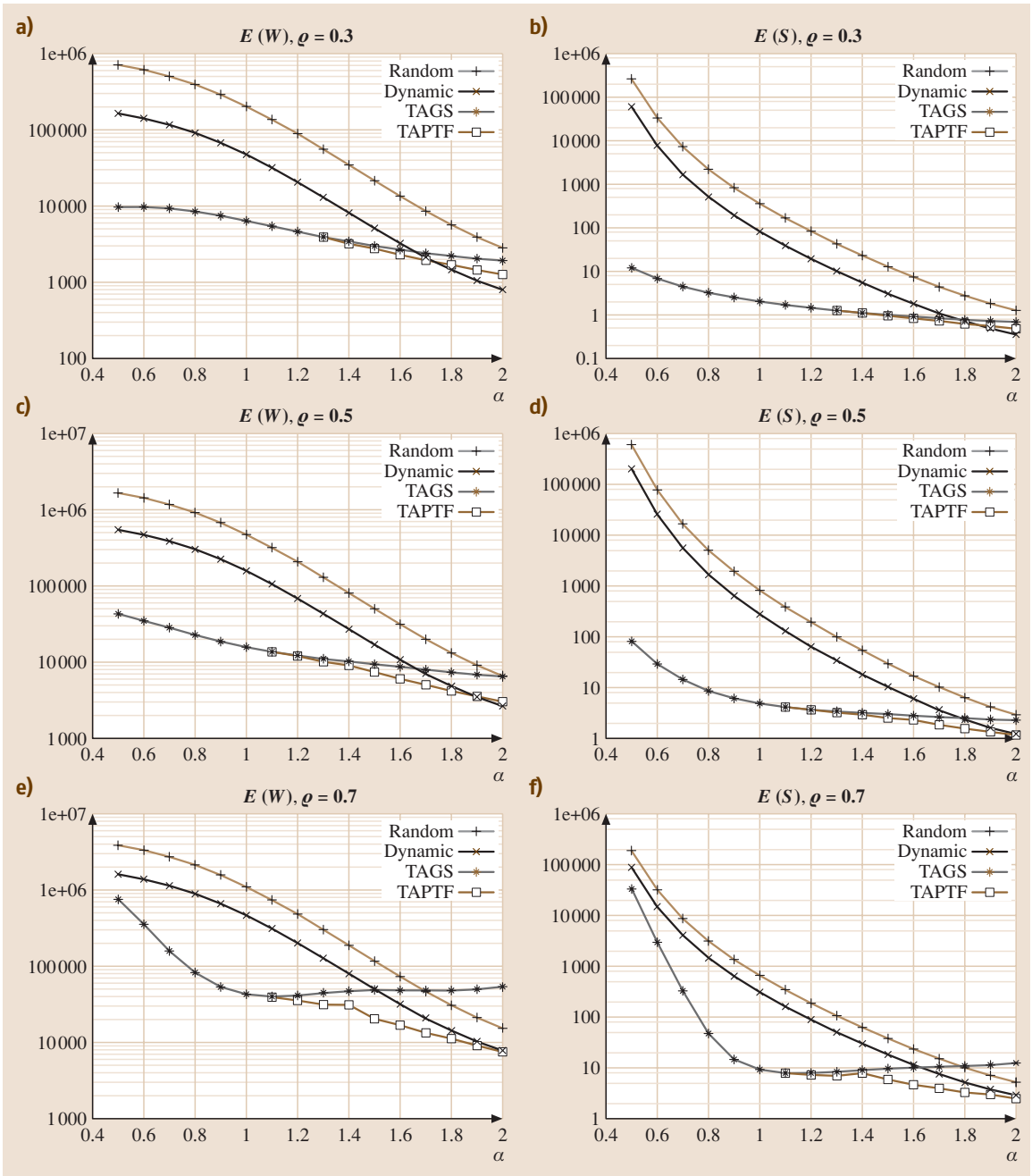

Fig. 52.13 Performance of a two-host distributed system with system load of 0.3, 0.5 and 0.7. The expected waiting time and slowdown are depicted under each load scenario

all tasks, they therefore contribute most to the overall system metrics, such as expected waiting time and slowdown. Recall that our system metrics depend on the second moment of the task size distribution; therefore, by reducing the variance experienced by the majority of tasks, size-based task assignment policies can signif-

icantly improve a system's performance over traditional techniques under heavy-tailed workloads. Next, we consider the performance of a distributed server cluster, where again our arrival is Poisson and our service time follows a bounded Pareto distribution. We set p (our largest task) to 10^7, and vary k (our smallest task) to keep the distributional mean, $E\{X\}$, fixed at 3000.

Figure 52.13 shows a comparison between the random, dynamic, TAGS and TAPTF policies in a two-host distributed system. The system load is varied, showing performance where $\rho = 0.3$ (low load), $\rho = 0.5$ (moderate load) and $\rho = 0.7$ (high load). The metrics used are the expected waiting time and mean slowdown.

As observed in Sect. 52.3.1, the performance metrics of both the random and dynamic policies are dependant on the variation of the task size distribution. As the variation increases (when α decreases) the expected waiting time and slowdown increase rapidly. Also, we note that the scale of improvement shown by the dynamic policy over the random policy decreases slightly as the system load increases. As the system load increases, there is a lower probability of a host being idle, even under the random policy. As such the dynamic policy has less scope for improvement.

Significantly, we can see an enormous improvement for the size-based task assignment policies, especially under conditions of high and extreme task size variations. These policies by their very nature reduce the variance of task sizes at each host, by partitioning the workload amongst each host. This has the effect of grouping similarly sized tasks together at the queues of each host, consequently reducing the variance at each host.

52.4 Active Queue Management for TCP Traffic

A description of how the TCP congestion control mechanism works, using a sliding congestion window to control the flow of packets through the system, is given in Sect. 52.0.3. The way in which the window size changes depends on the version of TCP used, the most common being the standard algorithms, slow start and congestion avoidance.

52.4.1 TCP Algorithms

Slow Start

Most versions of TCP that use the slow start algorithm are greedy and will attempt to take as large a share of the network resources as possible. The larger the window size a connection has, the larger the amount of resources it has. The purpose of slow start is to attempt to find the largest congestion window size a connection can have without causing too much congestion on the network. The following describes how slow start changes the window size from the commencement of a connection. In the algorithm described next, $W(a)$ refers to the window size after a acknowledgements.

1. The window size is set to one and a single packet is transmitted: $W(0) = 1$.
2. When an acknowledgement (ACK) for a packet arrives, the window size is increased by one and slides, allowing two new packets to be transmitted and acknowledging that the previous packet has been received correctly:
$$W(a+1) = W(a) + 1 \,.$$
3. When the window size reaches a slow-start threshold W_t (ssthresh), the slow-start phase ends and the congestion-avoidance phase begins.

Notice that the number of packets transmitted grows exponentially due to an increase in the number of ACKs received as the window size increases. For example, send one packet, receive one ACK, now send two packets and receive two ACKs, each of which can send a further two packets and so on.

Congestion Avoidance

Congestion avoidance usually commences after a slow-start packet has been lost; its purpose is to increase the window size slowly in an attempt to provide optimum utilization while preventing packet loss. The following are the steps of the congestion-avoidance algorithm:

1. After an ACK for one packet arrives, the window moves across one slot and one new packet can be transmitted.
2. After an ACK for every packet in an entire window arrives, the window size is increased by one, which allows two new packets to be transmitted. Therefore the window size changes according to (52.40):
$$W(a+1) = W(a) + \frac{1}{\lfloor W(a) \rfloor} \,, \qquad (52.40)$$

where $\lfloor x \rfloor$ refers to the *floor function*. For example, if $W(a) = 5$, then an arrival of an ACK would change the window to $W(a+1) = 5.2$ and five ACKs must arrive in total for the window to increase to 6.

3. Congestion avoidance ends when a maximum window size (MWS) is reached or when a packet loss occurs.

The above algorithm linearly increases the window size. The window size will only increase by one when all the ACKs from the window are received. Notice that to find the value of the window size at any a we must always take the floor of $W(a)$.

Retransmission Algorithms

In TCP packet loss is an indicator of congestion in the network and can occur in both slow start and congestion avoidance. Two common methods for detecting packet loss are fast retransmit and timeout.

Fast Retransmit. When the receiving end (client) acknowledges a packet it also indicates what the next required packet is. This will always be the packet that has the lowest sequence number. For example if the client received packets 1, 3 and 4 the next packet required is 2. Every time a new packet arrives packet 2 will be requested (in an ACK) until packet 2 arrives. These acknowledgements are called duplicate ACKs. Fast retransmit is based on the number of duplicate ACKs that are received which is usually three. If three duplicate ACKs arrive at the sender then fast retransmit will occur. This is only possible if at least three packets have been transmitted after the lost packet. Figure 52.14a shows the sequence of events in a fast retransmit where only one packet is lost. Notice packet 2 is transmitted after the third duplicate ACK for packet 2 arrives.

Timeout. Every time a packet is transmitted a timer is started, this timer will timeout, i. e. expire, after an estimated ACK arrival time. This estimated ACK arrival time is based on the mean and standard deviation of previous ACK arrival times. A timeout loss is detected when an ACK fails to arrive within the estimated arrival time and if three duplicate ACKs are *not* received. This means a timeout can only occur when a fast retransmit does not occur. Figure 52.14b shows the sequence of

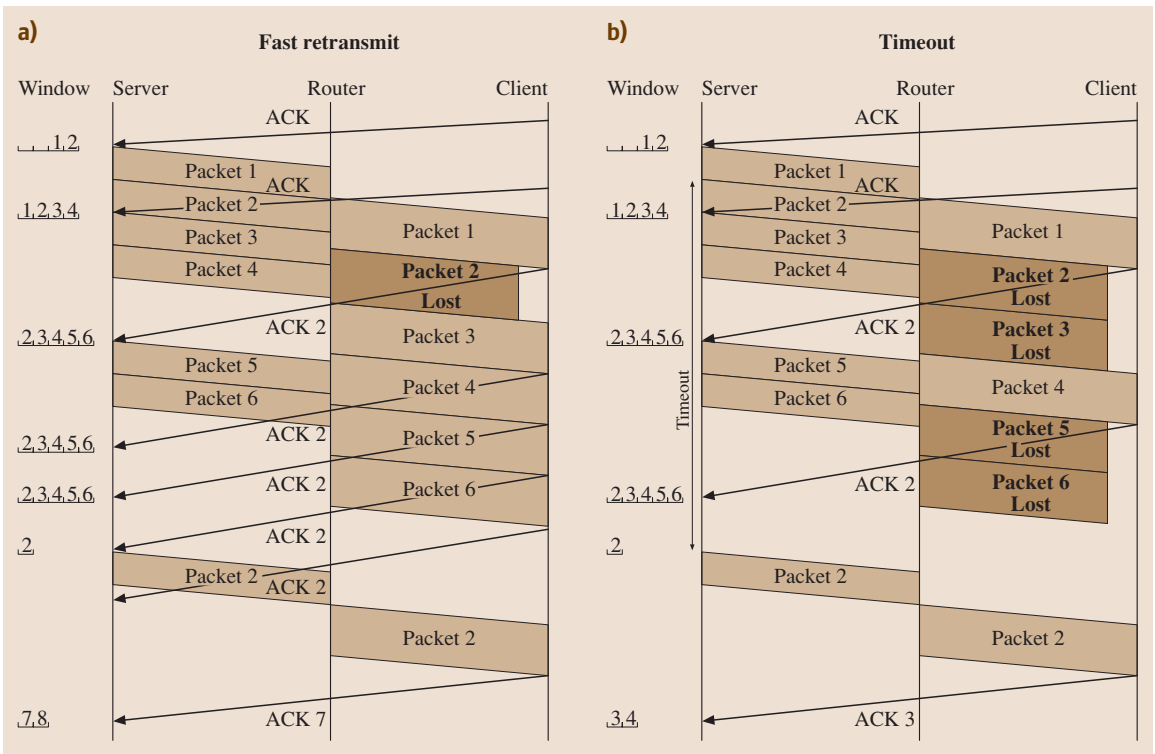

Fig. 52.14 Examples of fast retransmit and timeout

events for a timeout loss. Notice that only one packet is successfully transmitted after the first packet loss and only one duplicate ACK is received. Also notice that packets 5 and 6 are transmitted even though a packet loss has occurred.

52.4.2 Modeling Changes in TCP Window Sizes

Many models of TCP behavior have been proposed; most concentrate on finding the throughput of a TCP connections given some distribution of packet loss. Some models concentrate on short flows [52.47, 48] while other only consider long- or infinite-duration flows [52.49, 50]. A small number of models such as that of *Casetti* and *Meo's* [52.51] consider a mixture of long and short flows, which is the more realistic scenario. When short flows are modeled it is important to consider slow start because most short flows spend a majority of their time in slow start. For long flows, congestion avoidance is important because long flows spend most of their time in congestion avoidance. In [52.51], a model of the TCP congestion window size which represents both long and short finite-duration flows is presented.

In [52.52], the window size distribution is modeled as a Markov chain $\{U_k\}$, where k is the epoch of the change in window size. The state space of the process comprises four sets: active (N), idle (I), timeout (T) and fast retransmit (F), as shown in Fig. 52.15. N represents the set of states where data is available for TCP to transmit and members of this set are represented by the vector (W, W_t, N), where W is the window size, W_t the window threshold and W_M is the maximum received window size. The active states model the dynamic way the window size changes for both slow start and congestion avoidance. When a loss event occurs, the process makes a transition to one of two loss states, T and F, i.e to state (W, T) representing timeout has occurred, or (W, F) representing that fast retransmit has occurred. A transition to an idle set I models the situation when a connection has no data to send.

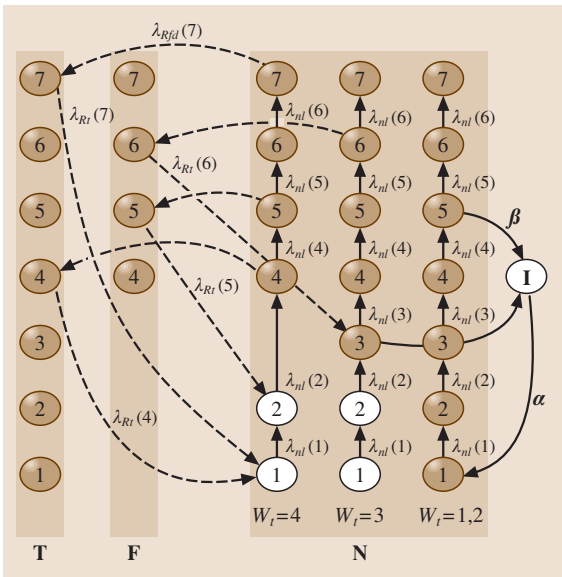

Fig. 52.15 Transitions between states in Markov model

From each of the active states N there are three different types of transition: timeout, fast retransmit and normal (i. e. no packets drop); each transition has a probability of occurrence of P_{Rt}, P_{Rf} and P_{nl}, respectively, which are used implicitly in calculating the transition rates between states. $U_k = (W, W_t, N)$ is an active transmitting state. All transitions from state U_k to U_{k+1} are summarized in (52.41) with their corresponding transition rates (the λs in the equations). For example if $W_M = 8$ and the current state is $(4, 2, N)$ then a transition to state $(5, 2, N)$ occurs if no packets are dropped. If a loss occurs then there is a transition to either state $(4, T)$, if it is a timeout, or $(4, F)$, if it is a fast retransmit. If there is no more data to send then there is a transition to state I.

$$U_{k+1} = \begin{cases} (2W, W_t, N)\lambda_{nl} & W < W_t \\ & 2W < W_t \\ (W_t, W_t, N)\lambda_{nl} & W < W_t \\ & 2W \geq W_t \\ (W+1, W_t, N)\lambda_{nl} & W_t \leq W < W_M \\ (W, T) & \lambda_{Rtd} \\ (W, F) & \lambda_{Rfd} \\ \mathrm{I} & \lambda_{I} \end{cases} \tag{52.41}$$

A timeout reduces the window size to one and the window threshold to $W/2$, which is half the window size, when a loss occurred. So from the timeout state $U_k = (W, T)$ a transition can occur to an active state $(1, \lceil W/2 \rceil, N)$ or the idle state if there is no more data to send, i. e.

$$U_{k+1} = \begin{cases} (1, \lceil W/2 \rceil, N) & \lambda_{Rt} \\ \mathrm{I} & \lambda_{I}\,. \end{cases} \tag{52.42}$$

Transitions from the fast-retransmit state $U_k = (W, F)$ are the same as the timeout state except the window size is reduced to half the loss window size instead of one, i. e.

$$U_{k+1} = \begin{cases} (\lceil W/2 \rceil, \lceil W/2 \rceil, N) & \lambda_{\mathrm{Rf}} \quad W > 3 \\ \mathrm{I} & \lambda_{\mathrm{I}} \,. \end{cases} \tag{52.43}$$

A transition from the idle state $U_k = \mathrm{I}$ to the initial active state $(1, \lceil W_M/2 \rceil, N)$ occurs with an arrival, i. e.

$$U_{k+1} = (1, \lceil W_{\mathrm{M}}/2 \rceil, N) \quad \alpha \tag{52.44}$$

where α is the arrival rate.

The stationary distribution π_{U} of $\{U_k\}$ can be obtained by solving the balance equations described by the transitions displayed in (52.41), (52.42), (52.43) and (52.44) in a similar way to that obtained for the MPDQ in Sect. 52.2.2. Note that the rates λ_{nl}, λ_{Rt} and λ_{Rf} are all functions of the window size W and must be defined from the packet-loss probability, loss distribution and delays of the network. A separate network model is required to define these parameters. Details of these derivations can be obtained from [52.52].

52.4.3 Modeling Queues of TCP Connections

The network that supports TCP connections is usually made up of a number of interconnected routers. Each router contains one or more buffers and a number of links. Bottlenecks can occur when a number of connections all attempt to utilize one link. Even though queueing can occur in any router, the bottleneck is usually most important because this is where the majority of queueing delay will occur. The most common type of queueing discipline assumed in routers are FCFS, the so called *droptail queues*. Recently active queue management (AQM) queues have been introduced specifically for TCP traffic. One of the most popular is the random early-detection queue (RED) [52.53]. In the following sections the FCFS droptail and RED queues will be described in more detail.

FCFS Droptail Queues

The most basic router queue on the internet is the droptail FCFS queue. A droptail queue has a finite buffer and drops packets if it is full. Packets arrive at the queue at the rate λ and are served at the rate μ. The arrival rate depends on the number of TCP connections connected to the queue and their maximum transmission rates. For example, if there are ten TCP connections with a maximum send rate of ten packets per second, then the maximum arrival rate would be one hundred packets per second. The departure rate μ is the number of packets that can be served on the outgoing link and depends on the bandwidth of the link. For example the outgoing link bandwidth may be 80 packets per second. The service time of each packet is deterministic because most packets of a TCP connection are the same size. The load of the network ρ is defined in Sect. 52.1.2. If the load is greater than one the network is considered congested and packets must be dropped from the queue. TCP aims to reduce the arrival rate of packets to the queue to a value close to one in an effort to minimize dropped packets and prevent packet retransmission, which adds further load.

An $M/D/1/K$ queue is a good model for approximating a queue of TCP connections because the arrival process of packets can be approximated as Poisson when there is a large number of sources, service time is deterministic and the queue has a finite capacity K. However, most of the operational parameters of an $M/D/1/K$ queue, such as average waiting time and drop probability P_K, i. e. the queue is full, cannot be obtained explicitly and the $M/M/1/K$ queue has been found to provide a good approximation [52.51]. As exhibited in Sect. 52.1.4, this simpler model has explicit equations for drop probability P_{K}, average waiting time $E(W_{\mathrm{q}})$ and average number of packets in the queue $E(L_{\mathrm{q}})$.

The average waiting time and packet drop probability together with the TCP send rate can be used to find the equilibrium point of a network using fixed-point analysis. The TCP send-rate equation finds the send rate (arrival rate to the queue) as a function of packet drop probability and average waiting time while the queue model finds the average waiting time and packet drop probability as a function of arrival rate.

Random Early Detection (RED)

As explained in previous sections, packet drops can control the send rate of a TCP connections. When the network contains droptail queues, the control of the TCP send rates is passive because only the queue length affects the packet drop probability. In active queue management the packet loss probability is controlled by the system administrator through various algorithms. These algorithms attempt to drop packets in strategic ways to control the send rate of TCP sources. One of the most common algorithms is RED [52.53].

RED attempts to control the rate of TCP traffic sources by dropping packets based on the average queue

Table 52.1 Some heavy-tail distributions

Description of files	α
Unix process CPU requirements [52.34]	1.0
Sizes of files transmitted over the internet [52.31, 32]	1.1 – 1.3
1998 World Cup web site file size distribution [52.36]	1.37

size, estimated using exponential weighted moving average (EWMA). It attempts to keep the average queue size low and reduce delay. A number of thresholds are used by the RED algorithm, which are defined as follows:

- $\min_{\text{th}}$ – the average queue size at which packets will start to be dropped. When the average queue size is less than $\min_{\text{th}}$ no packets are dropped.
- $\max_{\text{th}}$ – the average queue size at which all packets will be dropped if it is exceeded.
- $p_{\max}$ – the maximum probability of dropping a packet.

Using these threshold, RED calculates and sets the probabilities p_b and p_a, which it uses in selecting packets to drop. These probabilities are given by (52.45) and (52.46), which are functions of average queue sizes avg:

$$P_b(\text{avg}) = \begin{cases} 0 & \text{avg} < \min_{\text{th}} \\ 1 & \text{avg} > \max_{\text{th}} \\ p_{\max}\frac{\text{avg}-\min_{\text{th}}}{\max_{\text{th}}-\min_{\text{th}}} & \min_{\text{th}} \le \text{avg} \le \max_{\text{th}} \end{cases} \tag{52.45}$$

$$P_a = \frac{P_b}{1 - i P_b}\,, \tag{52.46}$$

where i is the number of packets since the last dropped or marked packet. As the average queue size increases so does the probability that packets are dropped randomly from the queue. The relationship between the average queue size and drop probability p_b is linear within the $\min_{\text{th}}$ to $\max_{\text{th}}$ range (with values from 0 to $p_{\max}$, respectively). When the average queue size is below $\min_{\text{th}}$ no packets are dropped and when it is above $\max_{\text{th}}$ all packets are dropped. Using average queue sizes allows the queue to grow up to its maximum size in order to accommodate erratic (bursty) traffic flows. We refer the reader to [52.53] for details regarding the choice between using p_b and p_a in marking and dropping packets.

The RED algorithm is also useful in providing differentiated services because it can control the rate in which different classes of packets enter the network. In the next section we will investigate how RED is applied to provide differentiated services.

Table 52.2 Scheduling variables

Variable	Description
x	Priority state (high $x = 1$, low $x = 2$)
L	The queue that obtains service
Q_x	The instantaneous queue length of priority x queue
W_x	The weight assigned to the priority x queue
R_1	Average goodput, i. e. the amount of packets that gets transmitted through a system (packets/second)
T_1	Goodput threshold of high priority traffic

52.4.4 Differentiated Services

Differentiated services allow differing traffic to be categorized into a number of different service groups that provide different levels of service. For example, some packets may require low delay, while for others high throughput may be important. In a network, service is provided to a packet at the router buffers. The router can then schedule the order in which packets are allowed access to the link. The two common scheduling algorithms are weighted round-robin (WRR) and priority scheduling, which we have discussed earlier in Sect. 52.2. Another popular method of differentiating service at a buffer is by exploiting TCPs congestion control mechanism to control the rate at which packets enter the network. Packets can be actively dropped to signal to the TCP sources to slow down and prevent the packets from entering the network in the first place. Algorithms that achieved this are weighted RED (WRED) and RED in/out (RIO). These and other mechanisms of differentiated services will be explained in the following sections. For simplicity only two groups of service (high and low priority) will be assumed when describing differentiated services. Table 52.2 gives a list of variables that will be used in the ensuing discussions.

Weighted Round-Robin (WRR)

Weighted round-robin is used in class-based queues (CBQ) as an extension to round-robin scheduling (cf. Sect. 52.3.1) where each priority has a separate first-in first-out (FCFS) queue with a weight (refer to Fig. 52.16). Each queue is assigned a number of slots depending on its weight and for each slot, a queue can transmit a packet of data. For example, say there is a total of three slots and the high-priority queue is assigned two of the three slots, the low-priority queue has the remain-

ing slot. The slots are served in a round-robin fashion by a single server who moves from queue to queue so that the high-priority queue gets twice as much service as the low-priority queue. Note that packets are also serviced in order of priority with two-high priority packets serviced then followed by one low-priority packet, and so on. This can be formally described as follows:

$$S = W_1 + W_2 \,,$$

$$L = \begin{cases} 1 & Q_2 = 0,\, Q_1 > 0 \\ 1 & Q_1 > 0,\, (a \bmod S) \le W_1 \\ 2 & Q_1 = 0,\, Q_2 > 0 \\ 2 & Q_2 > 0,\, W_1 \le (a \bmod S) < S \end{cases} ,$$

where S is the total weight and the variable a is discrete and increments by one after each service or when a queue is empty.

The benefit of WRR is that no class denies any other class services, the low-priority class will always get its allocated amount of bandwidth. This provides a degree of fairness to all classes much akin to that provided by the dual queue in a MPDQ.

Priority Queues

A priority queue has been discussed in previous sections but in the context of customers rather than packets. Here, a priority queue can be a single queue which rearranges packets based on their priority (as is assumed in Sect. 52.2) or it can be multiple queues which have a priority of services. A single-priority queue involves complex rearrangement of packets within the queue, which can be processor-intensive and hence difficult to implement in practice. For this reason, we will mainly concentrate on a priority CBQ where each queue serves packets of a single priority and the service of the queues is prioritized, i. e. a queue with high-priority packets are always served before that with low-priority packets. The only time a low-priority packet can be served is when the high-priority queue is empty. The scheduling algorithm is then simply:

$$L = \begin{cases} 1 & Q_1 > 0 \\ 2 & Q_1 = 0,\, Q_2 > 0 \,. \end{cases}$$

As we have seen, in priority queues if the amount of high-priority traffic is large the low-priority traffic can be starved of service completely. The major advantage of priority queues in packet networks is that high-priority packets have much lower delay because they are always served first [52.54].

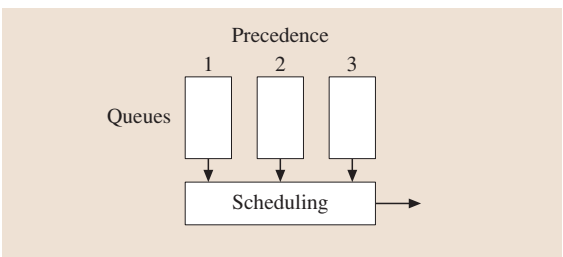

Fig. 52.16 A class-based queue

Flow-Based Quality of Service (QoS) with RED

A single flow is made up of many different types of packets, each of which can be given different levels of services. For example, a flow of size 60 packets could have 40 high-priority packets that must get to the other end and 20 low-priority packets that may get to the other end if there is available bandwidth. Weighted RED and RED in/out are a couple of ways that have been suggested to provide this differentiation within a flow.

Weighted RED (WRED). WRED [52.55] extends RED to allow different classes of packets to be treated differently. For example a high-priority class may have a lower probability of packet drop than a lower-priority class. WRED uses a single queue and maintains a single average queue size in exactly the same way as RED. It differs from RED by providing different $\min_{\text{th}}$, $\max_{\text{th}}$ and $p_{\max}$ for each class of packet. For example, WRED could be configured to have a larger $\min_{\text{th}}$ and $\max_{\text{th}}$ for high-priority traffic. This would make the high-priority packet-drop probability lower than that for low priority packets for all average queue sizes. A higher drop probability would mean that low-priority TCP sources would decrease their sending rate more than high-priority sources, thereby allowing more high-priority traffic through the link.

RED in/out (RIO). RIO [52.56] is similar to WRED except that it keeps a separate average queue length for the high-priority (*in*) packets and for the low-priority (*out*) packets. The reason different average queue sizes are used is to isolate the effect of *out* packets from *in* packets. For example if a single queue is used then a large load of *out* packets will increase the queue size and cause *in* packets as well as *out* packets to be dropped. By having a separate average queue size for *in* packets the number of *in* packets dropped is independent of the load of the *out* packets. RIO suffers the same prob-

lem as WRED in that when there is a large amount of *in* traffic it can starve low-priority *out* traffic of service.

Class-Based QoS with RED

A CBQ system can be combined with RED packet dropping to provide service differentiation between flows. Note this is a different level of service for an entire flow of packets rather than for individual packet. RED plus scheduling is an advantage over scheduling with FIFO because RED can control the source rate rather than just allocate desired bandwidth between different priority classes. For example if the high-priority traffic is allocated 1 Mbps out of 3 Mbps bandwidth, the high-priority load could be 2 Mbps. Rather than drop 1 Mbps of packets RED would attempt to adjust the load to 1 Mbps to fit the available bandwidth. The CBQ could use WRR or priority scheduling to serve each class of queue.

WRR RED. WRR RED simply combines WRR scheduling with RED. The WRR parameters are the same: each RED queue is allocated a number of service slots out of the total slots. Each RED queue can then use the same $\min_{\text{th}}$, $\max_{\text{th}}$ and $p_{\max}$, since the aim is to try to optimize the average queue length for the bandwidth allocated to the queue by WRR. WRR RED shares the advantages and disadvantages of WRR where high-priority packets are delayed by low-priority packets.

Table 52.3 DPRQ parameters

Parameter	Description
K	Maximum Buffer Size of both Queues
x	Priority (high $x = 1$, low $x = 2$)
D	Propagation delay
S_x	Number of priority x TCP sources
λ_x	Arrival rate of priority x packets
μ	Service rate of packets
P_{FS}	Probability of exceeding threshold given the current state is F and next state is S (above threshold S or $F = 1$, below threshold S or $F = 0$)
T_x	Goodput threshold of priority x queue
r_x	Average RED packet drop probability of priority x queue
g_x	Average RED packet no drop probability of priority x queue
d_x	Average total packet drop probability of priority x queue. RED plus queue full
q_x	Average queuing delay of priority x packets

Priority RED. Priority scheduling can also be combined with RED, just like WRR. Each RED queue is assigned a priority of services. For example if the highest-priority RED queue has packets in it, then it will always get service first. Priority RED faces the same advantages and disadvantages as the normal priority queueing discussed earlier. When there is a large amount of high-priority traffic it can starve low-priority traffic, with the advantage being that there would be a much lower delay for high-priority traffic.

Dynamic-Priority RED Queue (DPRQ)

Recently, we introduced the DPRQ [52.57], which extends the priority RED queue by adding a threshold to the amount of high priority traffic that is allowed service. The DPRQ scheduling algorithm allows a goodput i. e. successfully transmitted packets, threshold T_1 to be placed on high-priority traffic. The aim of the threshold is to prevent high-priority traffic from starving low-priority traffic, a drawback associated with the RED queues we have discussed earlier. The scheduling algorithm is as follows (Table 52.2):

$$L = \begin{cases} 1 & R_1 \le T_1,\ Q_1 > 0 \\ 2 & R_1 > T_1,\ Q_2 > 0 \\ 2 & Q_1 = 0,\ Q_2 > 0 \\ 1 & Q_1 > 0,\ Q_2 = 0 \end{cases} . \qquad (52.47)$$

When the average goodput of high-priority traffic R_1 is less than the threshold T_1 the high-priority queue will have priority in service. If R_1 increases beyond the threshold T_1 then it has exceeded its allowed goodput and the low-priority queue will be served with priority until R_1 is reduced. In this case R_1 naturally reduces because the high-priority queue is not being serviced. When there are no high-priority packets the low-priority queue will be serviced, if it is not empty. If the low-priority queue is empty then the high-priority queue is serviced. The average goodput R_1 is calculated over a specified time period using an exponential weighted average, just like that used to find the average queue length in the RED algorithm.

The DRPQ model and algorithm [52.57] will now be described in detail using the parameters which we have collected in Table 52.3.

As stated before in RED, the packet-drop probability depends on the average queue length. If the RED router is well configured the average queue length will not change a great deal over a large period of time. Over this period the average packet drop probability will be

Table 52.4 States of the DPRQ

Range	Description
$m = 0, n = 0$	Both queues empty
$m = 0, 0 < n < K$	High priority queue is empty
$m = 0, n = K$	High priority queue empty and the low priority queue full
$0 < m < K, n = 0$	Low priority queue empty
$m = K, n = 0$	Low priority queue empty and the high priority queue full
$0 < m < K, 0 < n < K$	General queue
$m = K, 0 < n < K$	High priority queue full
$0 < m < K, n = K$	Low priority queue full
$m = K, n = K$	Both queues full

P_b from (52.45). The probability of a packet not being dropped is therefore $g_x = 1 - P_b$, where x represents the priority of the queue.

Two possible states are defined for the threshold: either the threshold has been exceeded which is represented by a 1 or the threshold has not been exceeded, represented by a 0. The probability of the threshold changing from state F to state S is P_{FS}. For example, P_{01} is the probability, given that the threshold is currently not exceeded, that it will be exceeded in the next transition. Clearly, P_{FS} satisfy the following equations:

$$P_{00} + P_{01} = 1 \,,$$
$$P_{10} + P_{11} = 1 \,.$$

We assume that the number of packets served in any time interval has a Poisson distribution, which is a valid assumption if the number of packets traversing the queue is large, as it is in this case. Therefore, the goodput R_1 is also Poisson with mean $\mu_1 = S_1 g_1 \lambda_1$ and standard deviation $\sigma_1 = \sqrt{S_1 g_1 \lambda_1}$. If the mean is large, which it is here since λ and S_x are usually large, the cumulative normal distribution can be used to approximate the cumulative Poisson distribution, i. e. we approximate R_1 by a normal random variable X. Therefore, for $y \geq 0$

$$P(R_1 \leq y) \approx P(X \leq y) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{y - \mu_1}{\sigma_1 \sqrt{2}}\right)\right] , \qquad (52.48)$$

where $\operatorname{erf}(z) = \dfrac{2}{\sqrt{\pi}} \displaystyle\int_0^z \mathrm{e}^{-t^2}\,\mathrm{d}t$

is the error function. To approximate the probability that the threshold T_1 is not exceeded, simply set $y = T_1$ in (52.48) giving the following equation:

$$P_{00} = P_{10} = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{T_1 - \mu_1}{\sigma_1 \sqrt{2}}\right)\right] .$$

The queue length process is modeled using the Markov chain $\{V_k\}$ defined at the epoch of the k-th packet arrival to the queue. The states of $\{V_k\}$ are represented by the vectors (m, n, G), where m is the number of packets in the high-priority queue, n is the number of packets in the low-priority queue and G represents whether the threshold is exceeded in the high-priority queue: $G = 1$ signifies that the threshold is exceeded while $G = 0$ signifies that it is not. Table 52.4 indicates the possible range of states of V_k.

Different rate diagrams are needed for the different ranges shown in Table 52.4 but, due to lack of space, we will only show diagrams for the general range $0 < n < K$ and $0 < m < K$. We note that there are two sets of diagrams describing these transitions, one when $G = 0$ and another when $G = 1$. When the threshold has not been exceeded, the transitions from state $V_k = (m, n, 0)$ to V_{k+1} when $0 < m < K$ and $0 < n < K$ are described by (52.49) with corresponding transition rates:

$$V_{k+1} = \begin{cases} (m, n+1, 0) & g_2 \lambda_2 \\ (m+1, n, 0) & g_1 \lambda_1 \\ (m-1, n, 0) & P_{00}\mu \\ (m-1, n, 1) & P_{01}\mu \,. \end{cases} \qquad (52.49)$$

When the threshold has been exceeded, the transition from $V_k = (m, n, 1)$ to V_{k+1} when $0 < m < K$ and $0 < n < K$ are described by (52.50).

$$V_{k+1} = \begin{cases} (m, n+1, 1) & g_2 \lambda_2 \\ (m+1, n, 1) & g_1 \lambda_1 \\ (m, n-1, 0) & P_{10}\mu \\ (m, n-1, 1) & P_{11}\mu. \end{cases} \qquad (52.50)$$

Figure 52.17 shows the general transition-rate diagram. When the threshold is not exceeded the interactive

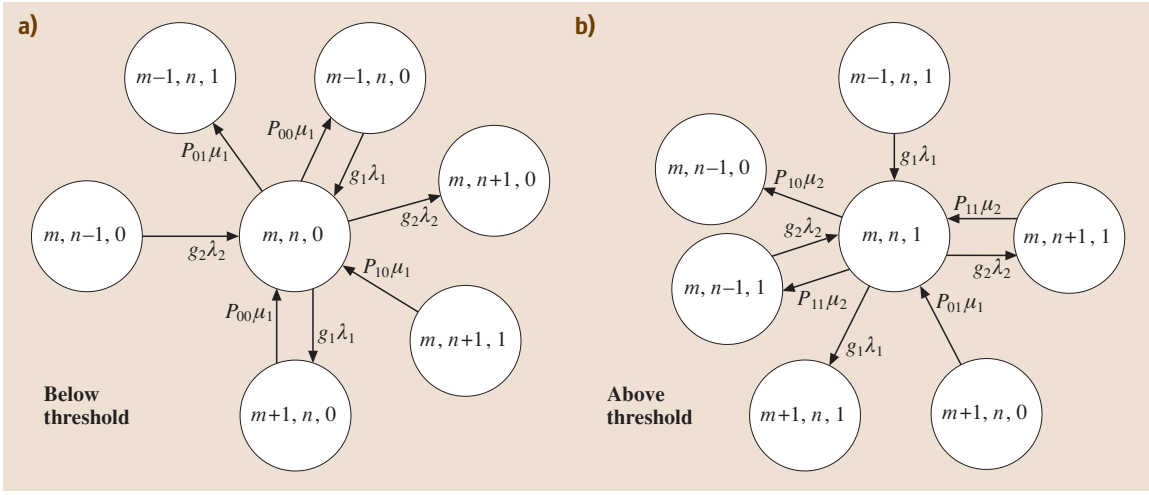

Fig. 52.17 DPRQ Markov process model ($0 < n < K$) and ($0 < m < K$)

queue has priority, therefore interactive packets are only served from states where the threshold has not been exceeded ($G = 0$ states). Similarly the low-priority queue is only serviced when the threshold has been exceeded therefore transitions can only occur from the low-priority queue when $G = 1$. Arrival of packets to either queue will cause the queue to increase in size.

Finally, extensive simulations based on the DPRQ and reported in [52.57] show general performance improvements, such as the reduction of packet loss, when compared to RED CBQ.

52.5 Conclusion

Modern communication systems, as exemplified by web server systems, continue to grow in complexity and level of sophistication. Queueing models have been extensively used to model and test new communication systems with the aim of improving QoS to users. In this chapter, we have discussed several approaches that have been advocated to reduce congestion and improve load balancing in these systems. We have considered both theoretical and practical aspects of analyzing and evaluating new systems, especially when the traffic flows are differentiated into different priority classes. The MPDQ discussed in Sect. 52.2 have been analyzed thoroughly, both in the theoretical and practical sense. However, a complete theoretical analysis of the network of multiple dual queues used in load balancing in Sect. 52.3 remains a challenge, as is the dynamic-priority RED queue (DPRQ) discussed in Sect. 52.4. All this is tied up with the feasibility of a comprehensive theoretical analysis of networks of queues with a specific type of node. Success in this venture would provide a unique opportunity to extend the range of problems that arise when designing new and reliable communication systems.

References

52.1 A. K. Erlang: Solution of some probability problems of significance for automatic telephone exchanges, Elektroleknikeren **13**, 5–13 (1917)

52.2 C. Palm: Intensitatsschwankungen im Fernsprechverkehr, Ericsson Technics **44**, 1–89 (1943)

52.3 M. A. Labrador, S. Banerjee: Enhancing application throughput by selective packet dropping, Proceedings of IEEE International Conference on Communications (ICC), Vancouver (1999) 1217–1222

52.4 W. Feng, D. D. Kandlur, D. Saha, K. G. Shin: Adaptive packet marking for maintaining end-to-end

throughput in a differentiated services internet, IEEE/ACM Trans. Network. **7**(5), 685–697 (1999)
52.5 S. G. Golestani: A self-clocked fair queueing scheme for broadband applications, Proceedings of the IEEE Infocom, Toronto (1994) 636–646
52.6 K. T. Chan, B. Bensaou, D. H. K. Tsang: Credit-based fair queueing (CBFQ), IEEE Electron. Lett. **33**(7), 584–585 (1997)
52.7 J. Jang, S. Shim, B. Shin: Analysis of DQLT scheduling for an ATM mMultiplexer, IEEE Commun. Lett. **1**(4), 175–177 (1997)
52.8 J. Wang, L. Yonatan: Managing Performance using Weighted Round Robin, IEEE Symposium on Computers and Communications **1**, 785–792 (2000)
52.9 D. Hayes, M. Rumsewicz, L. Andrew: Quality of service driven packet scheduling disciplines for real-time applications: looking beyond fairness, IEEE Infocom 1999 **1**, 405–412 (1999)
52.10 M. Shreedhar, G. Varghese: Efficient fair queuing using Deficit Round Robin, IEEE/ACM Trans. Network. **4**(3), 375–385 (1996)
52.11 R. Ranasinghe, L. Andrew, D. Hayes, D. Everitt: Scheduling disciplines for multimedia WLANs: Embedded round robin and wireless dual queue, Proc. IEEE Int. Conf. Commun. (ICC), ed. by K.-P. Estola, Helsinki (2001) 1243–1248
52.12 A. Bedford, P. Zeephongsekul: On a dual queueing system with preemptive priority service discipline, Eur. J. Oper. Res. **161**, 224–239 (2005)
52.13 T. Saaty: *Elements of Queueing Theory with Applications*. (Dover, New York 1961)
52.14 S. Asmussen: *Applied Probability and Queues* (Springer, Berlin Heidelberg New York 2003)
52.15 A. Allen: *Probability, Statistics and Queueing Theory: With Computer Science Applications* (Academic, New York 1990)
52.16 D. Gross, C. Harris: *Fundamentals of Queueing Theory* (Wiley, New York 1998)
52.17 N. Jaiswal: *Priority Queues* (Academic, New York 1968)
52.18 F. Kelly: *Reversibility and Stochastic Networks* (Wiley, New York 1979)
52.19 L. Kleinrock: *Queueing Systems 1: Theory* (Wiley, New York 1975)
52.20 L. Kleinrock: *Queueing Systems 2: Computer Applications* (Wiley, New York 1975)
52.21 M. L. Chaudhry, J. G. C. Templeton: *A First Course in Bulk Queues* (Wiley, Canada 1983)
52.22 D. Kendall: Stochastic processes occurring in the theory of queues and their analysis by the method of the embedded Markov chain, Ann. Math. Stat. **24**, 338–354 (1953)
52.23 J. D. C. Little: A simple proof of $L = \lambda W$, Oper. Res. **9**, 383–387 (1961)
52.24 K. L. Chung: *Markov Chains with Stationary Transition Probabilities* (Springer, Berlin Heidelberg New York 1960)
52.25 W. D. Kelton, R. P. Sadowski, D. A. Sadowski: *Simulation with Arena*, 2nd edn. (McGraw–Hill, New York 2002) p. 2
52.26 P. Zeephongsekul, A. Bedford: Waiting time analysis of the multiple dual queue with a preemptive priority service discipline, Eur. J. Oper. Res. (2006) (In press)
52.27 P. Zeephongsekul, A. Bedford: Analysis of the non-preemptive Multiple Priority Dual Queue, Department of Mathematics and Statistics Research Report **2** (2004)
52.28 M. Neuts: *Matrix-Geometric Solutions in Stochastic Models, An Algorithmic Approach* (John Hopkins Univ. Press, Baltimore 1981)
52.29 D. Wagner, U. Krieger: Analysis of a finite buffer with non-preemptive priority scheduling, Commun Stat.-Stoch. Models **15**, 345–365 (1999)
52.30 R. Wolff: Poisson arrivals see time averages, Oper. Res. **30**, 223–231 (1982)
52.31 M. Crovella, M. Taqqu, A. Bestavros: *Heavy-Tailed Probability Distributions in the World Wide Web* (Chapman Hall, New York 1998)
52.32 M. Crovella, A. Bestavros: Self-similarity in World Wide Web traffic: evidence and possible causes, IEEE/ACM Trans. Network. **5**(6), 835–846 (1997)
52.33 G. Irlam(1993): Available at http://www.base.com/gordoni/ufs93.html
52.34 M. Harchol-Balter, A. Downey: Exploiting process lifetime distributions for dynamic load balancing, ACM Trans. Comp. Sys. **15**(3), 253–285 (1997)
52.35 M. Arlitt, T. Jin: Workload characterization of the 1998 world cup web site, IEEE Network **14**, 30–37 (2000)
52.36 A. Iyengar, M. Squillante, L. Zhang: Analysis and characterization of large scale web server access patterns and performance, World Wide Web **2**, 85–100 (1999)
52.37 M. Harchol-Balter: Task assignment with unknown duration, J. ACM **49**(2), 260–288 (2002)
52.38 M. Harchol-Balter, M. Crovella, C. Murta: On choosing a task assignment policy for a distributed server system, J. Parallel Distrib. Comput. **59**(2), 204–228 (1999)
52.39 A. Silberschatz, P. Galvin: *Operating System Concepts*, 5th edn. (Addison–Wesley, Reading 1998)
52.40 M. Crovella, M. Harchol-Balter, C. Murta: Task assignment in a distributed system: improving performance by unbalancing load, Proceedings of ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Madison, USA 1998, ed. by S. Leutenegger (ACM Press, New York 1998) 268–269
52.41 M. Harchol-Balter, C. Li, T. Osogami, A. Scheller-Wolf, M. Squillante: Task assignment with cycle stealing under central queue, International Con-

ference on Distributed Computing Systems (ICDCS'03) **23**, 628–637 (2003)
52.42 R. W. Weber: On the optimal assignment of customers to parallel servers, J. Appl. Probab. **15**, 826–834 (1978)
52.43 R. D. Nelson, T. K. Philips: An approximation for the mean response time for shortest queue routing with general interarrival and service times, Perform. Eval. Rev. **7**(1), 181–189 (1978)
52.44 S. Sozaki, R. Ross: Approximations in finite capacity multiserver queues with Poisson arrivals, J. Appl. Probab. **13**, 826–834 (1978)
52.45 B. Schroder, M. Harchol-Balter: Evaluation of task assignmentpolicies for supercomputing servers. The case for load unbalancing and fairness, 9th IEEE Symposium on High Performance Distributed Computing , 211–220 (2000)
52.46 J. Broberg, Z. Tari, P. Zeephongsekul: Task assignment based on prioritising traffic flows, Lect. Notes Comp. Sci. **3544**, 415–430 (2005)
52.47 N. Cardwell, S. Savage, T. Anderson: Modelling TCP latency, Proc. IEEE Infocom, Isreal 2000, ed. by M. Sidi (IEEE, USA 2000) 1742–1751
52.48 T. Lakshman, U. Madhow: The performance of TCP/IP for networks with high bandwidth-delay products and random loss, IEEE/ACM Trans. Network. **5**(3), 336–350 (1997)
52.49 A. Kumar: A comparative performance analysis of versions of TCP in a local network with a lossy link, IEEE/ACM Trans. Network. **6**(4), 485–498 (1998)
52.50 J. Padhye, D. Firouiu, D. Towsley, J. Kurose: Modeling TCP reno performance: a simple model and its empirical validation, IEEE/ACM Trans. Network. **8**(2), 133–145 (2000)
52.51 C. Casetti, M. Meo: A new approach to model the stationary behaviour of TCP connections, IEEE Infocom 1999 **1**, 367–375 (2000)
52.52 P. Dimopoulos, P. Zeephongsekul, Z. Tari: Modeling the burstiness of TCP, Proceedings of the IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecomunication Systems, Netherlands 2004, ed. by D. DeGroot, P. Harrison (IEEE, USA 2004) 175–183
52.53 S. Floyd, V. Jacobson: Random early detection gateways for congestion avoidance, IEEE/ACM Trans. Network. **1**(4), 397–413 (1993)
52.54 S. Floyd, V. Jacobson: Link-sharing and resource management models for packet networks, IEEE/ACM Trans. Network. **3**(4), 365–386 (1995)
52.55 Weighted random early detection, Tech. Rep. (CISCO Corp., USA 2003)
52.56 D. Clark, W. Fang: Explicit allocation of best-effort packet delivery service, IEEE/ACM Trans. Network. **6**(4), 362–373 (1998)
52.57 P. Dimopoulos, P. Zeephongsekul, Z. Tari: Reducing the user perceived delay of interactive TCP connections using a dynamic priority approach, Proceedings of the 14th International Conference on Computer Communications and Networks (ICCCN), USA 2005, ed. by E. K. Park (IEEE, USA 2005) 421–427