3

Performance Analysis Basics

In this chapter we present basic methods for analysing, predicting and optimising the performance of networks. We start with the standard theory for networks, the queueing theory. After that, we address the network calculus, a relatively new theory for deterministic and stochastic queueing networks. Network calculus is on its way to becoming a system theory for queueing networks but being a new discipline still lacks wide-scale recognition, and there are still many issues to be solved. Finally, we give a primer of optimisation techniques that come in handy for optimising networks.

3.1 Queueing Theory

3.1.1 Introduction

Queueing theory is the main method for the analysis of networks. There is a large amount of literature about queueing theory, also because it can be applied to more areas than just computer networks. The canonical reference for queueing theory is still Kleinrock (1975, 1976). An up to date overview can be found in Bolch *et al.* (1998).

Queueing theory deals with stochastic queueing systems. It tries to answer questions like, for example, the mean waiting time of a packet in a queue or the mean utilisation of a router. The behaviour of the system in its equilibrium state is in the foreground of the analysis, since results of systems in transient states are relatively hard to get. The stochastics of the system lie in the arrival and service processes. For analytical reasons, it is often assumed that distributions for arrival and/or service processes are memoryless, even though there are some results for general distributions. In the last couple of years, progress has been made in the direction of more realistic arrival processes. Nonetheless, the core results of queueing theory are still heavily based on the memorylessness of the underlying distributions – an assumption that is very much in question for Internet traffic, see Section 5.2. This means that results obtained from queueing theory should not be trusted blindly. Despite this, queueing theory is still the most important method for analysing networks. We now discuss the basic queueing systems that are important for analysing networks. They are used in various places throughout the book.

3.1.2 Kendall's Notation

The basic queueing model is shown in Figure 3.1. Packets arrive randomly and are stored in a queue until they are processed by the server (or node or router). The arrival of packets is characterised by the *arrival process*, the service time by the *service process*. The order in which packets are processed is determined by the *service discipline*. There can be *multiple servers* for the same queue and the *buffer size of the queue* can be limited. Depending on how these five parameters are chosen, we determine different queueing systems. The parameters allow the short description of a queueing system using Kendall's notation. In Kendall's notation, a queueing system is described with five abbreviations

$$A/S/m/B - S$$

where A is the arrival process and S the service process. The following abbreviations are common for the arrival and service processes:

M (Markovian) A Markovian process has a Poisson distributed arrival of events (packets). A Poisson distribution has exponentially distributed interarrival times – that is the time between two packet arrivals when talking of the arrival process or the time to process a packet when talking of the service process. The probability distribution $A(t)$ is therefore $A(t) = 1 - \mathrm{e}^{-\lambda t}$ with λ as the only parameter. The expected interarrival time is $E(t) = 1/\lambda$ and the variance $\mathrm{Var}(t) = 1/\lambda^2$. To describe arrival processes, λ is typically used as a parameter and for service processes μ is used accordingly.

D (Deterministic) A deterministic 'distribution' has a constant value. Constant bit rate traffic could be characterised by a deterministic arrival process.

G (General) The general 'distribution' in Kendall's notation stands for a distribution where nothing except (in most cases) the mean and the variance is known.

Parameter m stands for the number of parallel servers. When trying to model a single router, m will typically be one. The maximum size of the buffer space available for the queue is described by B (counted in packets). Often, an infinite buffer size is assumed. This is represented by dropping parameter B from the notation. In an infinite queue, no packets will ever get dropped and the dropping probability becomes zero. At the same time, the queueing theory will raise a stability condition that has to be met so that the system's average queue length does not become infinite.

The last parameter S stands for the service discipline. The most important service disciplines are the following:

FIFO (or FCFS) The First In, First Out (aka First Come First Serve) service discipline is the default discipline and is assumed if the parameter S is not explicitly specified. Packets are served in the order in which they arrive.

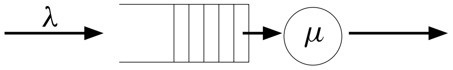


Figure 3.1 Queueing Model

PS (Processor Sharing) In a processor sharing service discipline, all packets in a queue receive service. The rate is split evenly among all waiting packets. A packet leaves the queue when it has acquired enough time slices.

PRIO (Priority) Each packet is assigned a priority and the server selects as the first packet that with the highest priority and serves it. If a pre-emptive priority discipline is assumed and a packet of a priority higher than the one currently being served arrives, then the service of the current packet stops and the higher priority packet is served. For a non-pre-emptive priority discipline, the service for the current packet is first fully finished until the high priority packet is being served. Non-pre-emptive priority scheduling is typically used for analysing communication networks.

A service discipline is called *work conserving* if it always serves a packet if there is at least one packet waiting in the queue. This means that as long as packets are waiting, the server is not idle. This can be assumed for the purpose of network analysis with queueing theory for all practical cases.

The most famous queueing system is the M/M/1 queueing system. It has a Markovian arrival process (interarrival times are exponentially distributed) and a Markovian service process (service times are exponentially distributed). There is only one server; it has infinite buffer for the queue and serves in First in, First out (FIFO) order. This system is discussed below. Afterwards the M/M/1/B system is discussed; it is similar to M/M/1 but only has a finite buffer size. This means that packets can get lost, and we can derive an average dropping probability. Finally, we look at a system where the service time is no longer Markovian. But first, we look at one of the most important laws in queueing theory, Little's law.

3.1.3 Little's Law

Little's law was first described in Little (1961). It is a very general law that holds even for G/G/1 queues and all service disciplines that are work conserving. Let $E(S)$ describe the *expected queueing delay* of a packet in the system, also called *sojourn time*. $1/\mu$ is the average service time; $1/\lambda$, the interarrival time; and $E(W)$, the waiting time. The *waiting time* describes the time in queue only (not in the complete system) so that $E(S) = E(W) + 1/\mu$. Let $E(L)$ be the expected number of packets waiting in the queue and $E(N)$ be the number of packets in the complete system (queue and server).

Little's law says: The average number of packets in the queue is equal to their average arrival rate, multiplied by their average waiting time

$$E(L) = \lambda E(W) \tag{3.1}$$

For systems without loss, this can be easily applied to the whole system as well[1].

$$E(N) = \lambda E(S) \tag{3.2}$$

[1] For lossy systems, the throughput and not the arrival rate would have to be used.

3.1.4 M/M/1 Queueing Systems

The M/M/1 queue has an infinite buffer space and uses a FIFO service discipline. The arrival process has exponentially distributed interarrival times with an average interarrival time of $1/\lambda$. The service process has exponentially distributed service times with an average service time of $1/\mu$. The arrival and service *rates* are accordingly λ and μ.

The state of the system can be described with the Markov chain depicted in Figure 3.2 (a). The nodes represent the number of packets k in the system, the arrows represent the state transitions.

We analyse the system in steady state; that means in a state where the system is stable and the probabilities do not change over time. The probability that there are k packets in the system is denoted by p_k. The sum of all state probabilities p_k has to be 1

$$\sum_{k=0}^{\infty} p_k = 1 \tag{3.3}$$

For the steady state, the flow in and out of state 0 has to be equal

$$\mu p_1 = \lambda p_0 \tag{3.4}$$

The same holds true for all other states k

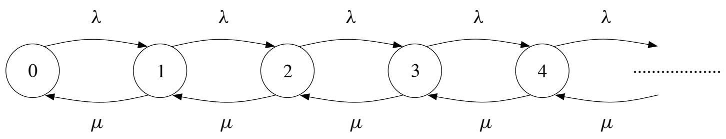
$$\lambda p_{k-1} + \mu p_{k+1} = \lambda p_k + \mu p_k \tag{3.5}$$

Equation (3.5) can be reformulated as

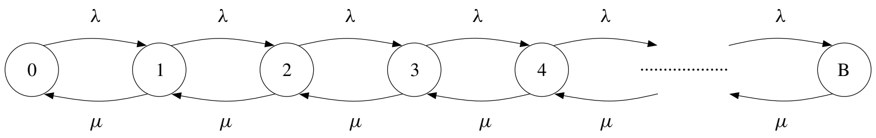
$$p_k = \left(\frac{\lambda}{\mu}\right)^k p_0 \tag{3.6}$$

Combining (3.3), (3.4) and (3.6) leads to

$$p_0 = 1 - \frac{\lambda}{\mu} \tag{3.7}$$



(a) Infinite Markov Chain of the M/M/1 Queue



(b) Finite Markov Chain of the M/M/1/B Queue

Figure 3.2 Markov Chains

With (3.6) and (3.7), the system can be described. Some other metrics are useful as well. The *utilisation* ρ of the system is defined as the fraction of time when there is a packet in the system. It is

$$\rho = 1 - p_0 = \frac{\lambda}{\mu} \tag{3.8}$$

The expected number of packets in the system $E(N)$ is given by

$$E(N) = \sum_{k=0}^{\infty} k p_k = \frac{\rho}{1-\rho} \tag{3.9}$$

if the condition $\rho < 1$ is met. This condition is the stability constraint for the M/M/1 queue. The expected queueing delay (sojourn time) $E(S)$ can be found with Little's law (3.2)

$$E(S) = E(N)/\lambda = \frac{1/\mu}{1-\rho} \tag{3.10}$$

The delay is depicted in Figure 3.3 as a function of the utilisation ρ.

3.1.5 M/M/1/B Queueing Systems

The M/M/1/B queue is similar to M/M/1 except for the limited buffer size B. Its Markov chain is shown in Figure 3.2 (b). The state probabilities can be derived the same way as before which yields for $\rho = \lambda/\mu$

$$p_0 = \frac{1-\rho}{1-\rho^{B+1}} \tag{3.11}$$

$$p_k = p_0 \rho^k \qquad 1 \le k \le B \tag{3.12}$$

Please note that for the M/M/1/B queue the M/M/1 stability condition $\rho < 1$ is not necessary; the system is stable even for $\rho > 1$.

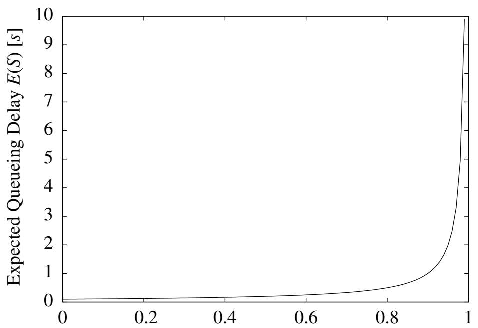


Figure 3.3 Delay of the M/M/1 Queue as Function of the Utilisation with $1/\mu = 0.1s$

The expected number of packets in the system $E(N)$ is

$$E(N) = \sum_{k=0}^{B} kp_k = \frac{\rho}{1-\rho} - \frac{B+1}{1-\rho^{B+1}}\rho^{B+1} \tag{3.13}$$

With Little's law (3.2) follows for the queueing delay (sojourn time)

$$E(S) = E(N)/\lambda = \frac{1}{\mu}\left(\frac{1}{1-\rho} - \frac{B+1}{1-\rho^{B+1}}\rho^{B}\right) \tag{3.14}$$

The loss probability p is the probability that an arriving packet finds the system full, which means that the system is in state p_B

$$p = p_B = \frac{(1-\rho)\rho^B}{1-\rho^{B+1}} \tag{3.15}$$

The loss probability and queue length of the M/M/1/20 queue are depicted in Figure 3.4.

3.1.6 M/G/1 Queueing Systems

The M/G/1 Queue is a generalisation of the M/M/1 queue where the service time follows an arbitrary distribution for which the mean value $E(x)$ and the standard deviation $\mathrm{Var}(x)$ are known.

The exponential interarrival process with parameter λ of the queue is a memoryless process. From this follows an important property of the M/G/1 queue, called PASTA. This stands for **P**oisson **a**rrivals **s**ee **t**ime **a**verages. It means that a new packet arriving at the queue sees exactly the same statistics of the number of packets in the system as when looking at the system at any random time in steady state. This does not hold for other arrival processes where the system can have different properties for different random points.

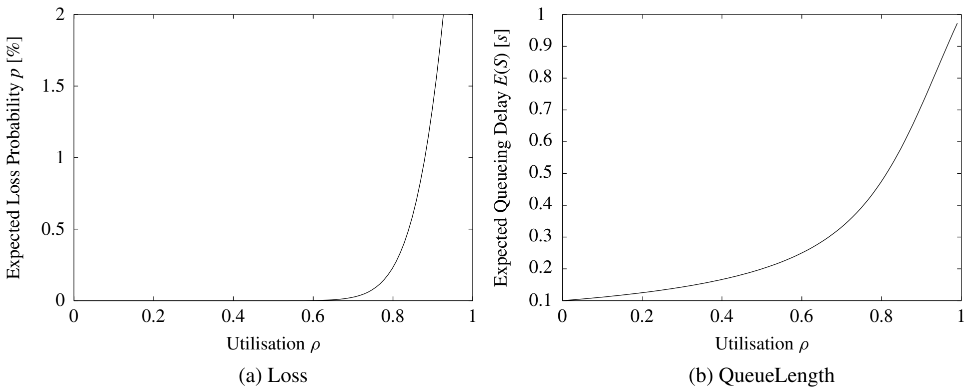


Figure 3.4 Loss Probability and Queueing Delay of an M/M/1/B Queue with $B = 20$

We define the utilisation ρ for this queue as $\rho = \min\{1, \lambda/E(x)\}$. The expected number of packets in the system $E(N)$ and the expected queueing delay (sojourn time) $E(S)$ for the M/G/1 queue are given by the so-called Pollaczek–Khinchin mean value formulas

$$E(S) = E(x)\left(1 + \frac{\rho(1 + C_v^2)}{2(1 - \rho)}\right) \tag{3.16}$$

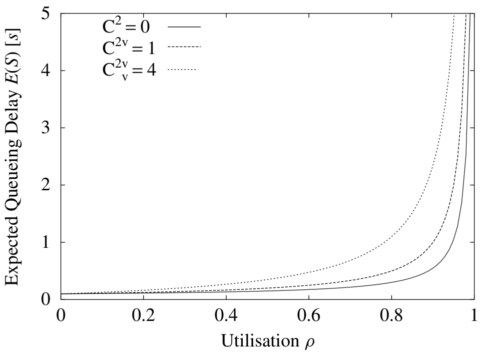
$$E(N) = \rho + \frac{\rho^2(1 + C_v^2)}{2(1 - \rho)} \tag{3.17}$$

where C_v^2 is the squared coefficient of variation $C_v^2 = \frac{\mathrm{Var}(x)}{E(x)^2}$. One can immediately see that the M/G/1 queue gets unstable as the utilisation ρ approaches 1. Since $E(x) = 1/\lambda$ and $\mathrm{Var}(x) = 1/\lambda^2$ for the exponential distribution, the Pollaczek–Khinchin mean value formulas simplify to (3.9) and (3.10) for the M/M/1 queue.

From the Pollaczek–Khinchin mean value formulas it follows that the squared coefficient of variation C_v^2 of the service time distribution has a strong influence on the expected delay and queue length. This is visualised in Figure 3.5. The lowest delays can be achieved when C_v^2 is zero, that is, when the service time is deterministic (constant). As a side remark, this is a valid assumption for ATM networks because of ATM's fixed cell size. For IP routers, the service time can be modelled as a constant processing overhead plus the time to put the packet on the outgoing link which is proportional to the packet size. The packet size distribution shows a few clear peaks and can be used as an input, see for example, Appendix D.2.

3.1.7 Other Queueing Systems

There are countless works on more advanced queueing systems. Let us cite one interesting example in Boyer *et al.* (2003). Here, a heavy-tailed M/G/1-PS (processor sharing) queue is being analysed under certain given assumptions. Heavy tailed refers to the service times, which decline slower than exponential. The most important constraint here is that impatient users are considered as well. This is justified by the assumption that the rates



(a) Queue Length

Figure 3.5 Expected Queueing Delay of an M/G/1 Queue for Different Coefficients of Variation

of single users decrease when many users are active at the same time, and some will get impatient and cancel their transfer. The analysis of this queue is used to connect the access control of elastic data flows and the number of the impatient users.

3.1.8 Queueing Networks

The modelling of entire *networks* of queueing systems is very complex compared to the single queueing systems discussed above. On the basis of the highly simplified assumptions, such as the memorylessness of all underlying stochastic systems, a relatively simple form for the probability of the state of the whole systems arises. It results as the product of marginal distributions of the single queueing systems; hence, the outcome is the so-called *network in product form.* Yet, a loosening of the strict assumptions, for example, the introduction of priorities or not requiring memoryless service times, will quickly lead to systems that are analytically not manageable anymore. Then, only numerical methods can help calculating the probabilities for the states of the systems. Numerical methods provide only approximations, however, and quickly reach calculation limits because of their recursive nature. Here, we refer to Whitt (1995) and Kouvatsos and Denazis (1994).

An expansion of queueing networks results from the introduction of negative packets, see Gelenbe (1993). With the arrival of a negative packet at a queue, a positive packet is erased; hence both the negative and positive packets disappear from the system. This expansion is relevant for the Internet, as for example RED (Random Early Detection) can be modelled with it.

Besides the other more general references mentioned above, Robertazzi (2000) provides an overview of queueing networks.

3.1.9 Conclusions

In conclusion, queueing theory is an analytical approach and it has a more locally oriented scope. Although there are queueing systems with feedback, they are typically not used in the context of communication networks. Queueing theory is mainly a model for explanations and is less suitable for forecasts and decisions than the network calculus, because it usually works with mean values.

3.2 Network Calculus

For performance analysis based on worst-case assumptions, the *traditional network calculus* offers a mathematical framework. It is a theory for deterministic queueing models. The Intserv guaranteed service (see Section 6.2.2.4) is a service based on network calculus considerations.

The network calculus is based on min-plus algebra (see Baccelli *et al.* (1992)), i.e. an algebra in which the operations + and * of conventional algebra are substituted by the calculation of the minimum and +. The beginnings of this theory can be traced back to Cruz (1991). It was further developed and described in detail by Le Boudec and Chang, among others, in Le Boudec and Thiran (2001) and Chang (2000). Chang elaborates mainly on the filter theory.

3.2.1 Basics

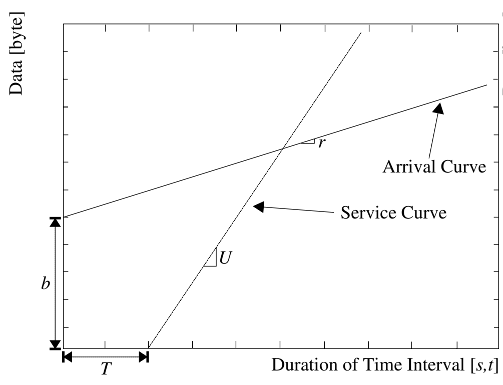
In the analysis of network behaviour, the network calculus focuses on the contemplation of worst-case characteristics of a data flow and thus delivers deterministically applicable statements. This is usually based on the assumption that guarantees can be based on a traffic description and admission control and a bound for the system behaviour.

The details of the traffic description are abstracted with the help of the so-called *arrival curves:* Assume that the total amount of transmitted data of a traffic flow over time is described by function $R(t)$. $R(t)$ is obviously wide-sense increasing. An arrival curve $\alpha(t)$ indicates the maximum amount of traffic that is permitted *over all possible time intervals*. Examples for arrival curves are token buckets and TSpecs. A *token bucket* is an algorithm used to describe shaped traffic. Assume a bucket that is filled with tokens at a rate r. The bucket can store no more than b tokens; b is called the bucket size. For transmitting a packet a token is taken from the bucket. If the bucket is empty, no packet may be transmitted. A *TSpec* is an extended token bucket that is used to specify traffic in Intserv; see Section 6.2.2.4.

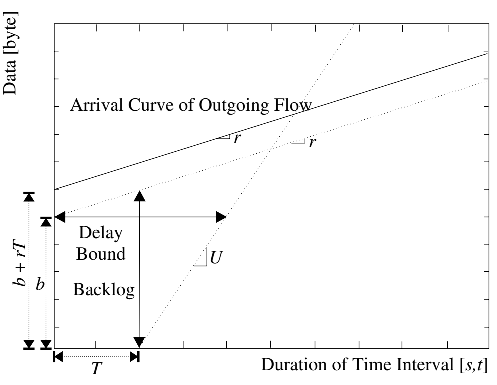
A token bucket arrival curve $\alpha(t) = rt + b$ is depicted in Figure 3.6 (a); it has to be interpreted in the following way: The y-axis shows the amount of traffic in bytes or packets. The x-axis represents the duration $t - s$ of every possible time interval $[s, t]$ since the beginning of the traffic flow. For every time interval, the maximum amount of traffic that is allowed according to the traffic description is given by the arrival curve. This means that if we select any point of time s of the lifetime of the traffic flow R, the amount of data that can be sent up to time t ($t \geq s$) is limited by $\alpha(t)$:

$$\alpha(t - s) \geq R(t) - R(s)\ \forall s \leq t \tag{3.18}$$

For the token bucket in Figure 3.6 (a), a traffic flow can send a traffic burst of maximum size b at a point in time s but in the long run the average rate will not exceed r.



(a) Arrival and Service Curve



(b) Results

Figure 3.6 Network Calculus Example

The second basic element of network calculus are *service curves*. A service curve describes the scheduling behaviour of a node on an abstract level. In network calculus, it is assumed that a node (or system) S allocates capacity to flow $R(t)$ to offer guarantees. These guarantees are expressed by the service curve $\beta(t)$. An example service curve is depicted in Figure 3.6 (a). It is a rate latency service curve

$$\beta(t) = \begin{cases} 0 & t \leq T \\ U \cdot (t - T) & t > T \end{cases}$$

the system imposes a delay of up to T and serves the flow with a long-term average rate of at least U.

A crucial result of network calculus is that the output (outgoing flow) $R^0(t)$ of a system (node) S is given by

$$R^0 \geq R \otimes \beta \tag{3.19}$$

The $\otimes$ operator is the min-plus convolution, which is defined as

$$(R \otimes \beta)(t) = \inf_{s|0 \leq s \leq t}\{R(s) + \beta(t - s)\} \tag{3.20}$$

If R is bounded by its arrival curve α, the arrival curve α^0 of R^0 can be calculated with the min-plus deconvolution $\oplus$

$$\alpha^0 = \alpha \oplus \beta \tag{3.21}$$

The min-plus deconvolution is given by

$$(\alpha \oplus \beta)(t) = \sup_{s|s \geq 0}\{\alpha(t + s) - \beta(s)\}$$

Another crucial result of network calculus is that the service curve β_{series} of two nodes in series is the min-plus convolution of the two service curves β_1 and β_2 of the single nodes.

$$\beta_{series} = \beta_1 \otimes \beta_2 \tag{3.22}$$

3.2.2 Example

Let us assume that a traffic flow $R(t)$ is bounded by the token bucket arrival curve $\alpha(t)$ and served in a node with the rate latency service curve $\beta(t)$ of Figure 3.6 (a). Then the arrival curve of the outgoing flow $R^0(t)$ is

$$\begin{aligned} \alpha^0(t) &= \sup_{s|s \geq 0}\{\alpha(t + s) - \beta(s)\} \\ &= b + rt + rT \end{aligned}$$

The result is depicted in Figure 3.6 (b).

The *backlog* describes the amount of buffer necessary in a system. The backlog is bounded by the vertical deviation between the arrival and service curves. The maximum backlog B for this example is shown in Figure 3.6 (b); in this case it is $b + rT$.

The network calculus also helps in determining a *delay bound* for traffic going through system S. This is very helpful for delay bounded services like the Intserv guaranteed service. The delay bound is given by the maximum horizontal deviation between the arrival and service curves. It is also depicted in Figure 3.6 (b). The delay bound is $T + b/U$ in this example.

Please note that the backlog and delay bound depend on the steepness of the arrival and service curves; they are not necessarily positioned as depicted in Figure 3.6 (b).

3.2.3 Conclusions

The traditional network calculus is an analytical approach that provides a model for the explanation of worst-case behaviour of data flows and can be used for forecasts and decisions. If all the calculations are done using the maximum traffic possible, one faces the risk of allocating the resources too conservatively and thereby causing a bad utilisation ratio. This effect amplifies with the aggregation of data flows. Therefore, one trend in network calculus research is the development of a statistical network calculus that could help at least partly in handling this problem; see the following text.

Since in network calculus it is assumed that the traffic meets a certain limit, feedback is normally not taken into account. Dynamics in terms of interaction with other data flows are also limited. In general, network calculus is only applicable to a limited number of use cases.

3.2.4 Outlook

The traditional network calculus is a system theory for deterministic queueing systems. The traditional network calculus is good for systems that give deterministic guarantees. Statistical guarantees, however, have a broader applicability and are more attractive for Internet Network Service Providers that wish to achieve a high utilisation of their network. Therefore, the traditional network calculus is currently being extended towards a statistical network calculus. Statistical network calculus (see e.g. Boorstyn *et al.* (2000)) is based on the assumption that an arrival curve – called effective envelope here – will be met only with a certain probability. Works on a statistical network calculus go back as far as Chang (1994) and Yaron and Sidi (1993). Lately, however, statistical network calculus is receiving much more attention. In Burchard *et al.* (2002), an effective service curve is introduced, which allows for the calculation of an output envelope with the convolution of effective envelope and effective service curve. The statistical network calculus is a promising field of research as it could develop a system theory for statistic queueing systems also. For the state of the art, we refer to Burchard *et al.* (2002); Ciucu *et al.* (2005); Fidler and Recker (2005); Fidler *et al.* (2005); Jiang and Emstad (2005a,b).

Another promising extension of the network calculus is that towards a calculus for sensor networks, see for example Schmitt and Roedig (2005).

3.3 Optimisation Techniques

3.3.1 Introduction

Queueing theory and network calculus allow us to analyse, understand, and predict the basic behaviour of nodes and smaller networks. In this section, we look at some basic optimisation techniques. Many of the problems INSPs face can be seen as *optimisation problems*: decisions have to be made, for example, about how and when the network capacity will be expanded. We call the parameters to decide upon as *decision variables* or in short just *variables*. The decision depends on many *input parameters* that are given or that have to be researched before the decision can be made. We call the input parameters shortly *parameters*. A certain function of these parameters and variables is to be maximised or minimised, for example, the costs of the network expansion. This function is called the *objective function*. Often there are certain restrictions that have to be observed, for example, that the traffic demand is sufficiently satisfied at all times. They are expressed as functions of the variables and parameters and are called *constraints.*

We place the focus on linear programming (LP) models and (mixed) integer programming models here. These models are used throughout this book to model various problems important for INSPs. There are many other optimisations problems and solution methods that we cannot present and discuss here owing to space limitations. We refer to the literature on operations research for more details. A good standard introductory book is Hillier and Lieberman (2001) that also forms the basis of this chapter.

In the next section, we look at how optimisation problems can be modelled mathematically. After that we discuss the standard methods used for solving these problems exactly.

3.3.2 Modelling Optimisation Problems

The standard approach to solving optimisation problems is to identify the exact problem, the variables, constraints, objective function, and parameters first. Then the objective function and constraints are formulated as a mathematical model. Certain parameters might have to be researched and often some simplifying assumptions are necessary to obtain the functions. Then the problem is solved, typically with the standard methods we discuss below. To increase confidence in the model and to find potential mistakes, often some small problems are solved first and the solutions are analysed carefully. If the confidence in the model is high, then the actual problem is solved. Because input parameters can be uncertain or incorrect, a sensitivity analysis is often performed after obtaining a solution to find how sensitive the solution is to changes of input parameters before the solution is applied.

Let us model a very simple optimisation problem now. The problem is very similar to the one in Hillier and Lieberman (2001), Chapter 3, in case the reader is interested in a more detailed discussion. It is an LP model:

- Assume that a company wants to produce two new products X_1 and X_2 without changing the already existing production line. Each batch of product X_1 makes a profit of \$3,000 and each batch of X_2 makes a profit of \$5,000. The company has three different machines, A, B and C that have different amounts of production time left per week. Producing the different products needs different production time per batch on all the three machines. Table 3.1 lists the production time for the different products and

Table 3.1 Production Time per Batch

Machine	Production Time per Batch		Total Available Production Time
	X_1	X_2	
A	1	0	4
B	0	2	12
C	3	2	18

the total available production time left on the machines. The goal is to maximise the total profit.

- To model this problem mathematically, we first introduce the decision variables. Variable x_1 describes the number of batches that is produced of product X_1 and variable x_2, the number of batches of product X_2. This helps us in formulating the objective function as a mathematical function: the total profit P (in thousand dollars) has to be maximised

$$\text{Maximise } P = 3 \cdot x_1 + 5 \cdot x_2 \tag{3.23}$$

Next the constraints have to be formulated. The constraints are given by the available production time of the different machines. As there are three machines, this results in the following three constraints

$$1 \cdot x_1 + 0 \cdot x_2 \leq 4 \tag{3.24}$$

$$0 \cdot x_1 + 2 \cdot x_2 \leq 12 \tag{3.25}$$

$$3 \cdot x_1 + 2 \cdot x_2 \leq 18 \tag{3.26}$$

Finally, we have to constrain the variables x and y to non-negative values to avoid the otherwise possible absurd results like produce -3 times product X_1 and 5 times X_2. The non-negativity constraints are

$$x_1,\ x_2 \geq 0 \tag{3.27}$$

The complete optimisation problem is now given by (3.23) to (3.27). If we abstract from the parameter values and replace the given number of products and machines with indices p and m, the general optimisation problem can be specified as in Model 3.1.

This optimisation problem is a *linear programming problem*, because all functions are linear and the variables are continuous. The word programming does not refer to computer programming but is used as a synonym for planning.

One potential problem of modelling the above-mentioned problem as an LP problem is that the variables are continuous. For example, the solution $x_1 = 3$ and $x_2 = 4.5$ is a valid solution of the problem mathematically. However, selling half a product might be impossible in reality. If that is the case, the optimisation problem has to be reformulated:

- Variables x_1 and x_2 have to be forced to integer values. This can be represented in the mathematical formulation by replacing (3.27) with

$$x_1,\ x_2 \in \{0, 1, 2, 3, \ldots\} \tag{3.28}$$

and (3.32) with

$$x_p \in \{0, 1, 2, 3, \ldots\} \ \forall p \tag{3.29}$$

The resulting problem is no longer an LP problem but an *integer programming problem*. You will see next that integer programming problems are much harder to solve than LP problems. In some optimisation problems, integer variables and continuous variables are mixed. These problems are called *mixed integer programming problems* (MIP) and generally the same methods used for pure integer programming problems can be used for these.

Model 3.1 Example Optimisation Problem

Indices	
$m = 1, \ldots, M$	Index for the different machines
$p = 1, \ldots, P$	Index for the different products
Parameters	
M	Number of machines
P	Number of products
t_{pm}	Production time of product p on machine m
T_m	Available production time on machine m
w_p	Profit for one batch of product p
Variables	
x_p	Number of batches produced of product p

$$\text{Maximise } \sum_p w_p x_p \tag{3.30}$$

subject to

$$\sum_p t_{pm} x_p \leq T_m \qquad \forall m \tag{3.31}$$

$$x_p \geq 0 \qquad \forall p \tag{3.32}$$

3.3.3 Solving Optimisation Problems

3.3.3.1 Linear Programming Problems

Graphical Solution The simple optimisation problem shown in the preceding text can be solved graphically as shown in Figure 3.7. The two decision variables x_1 and x_2 form the axes. Figure 3.7 (a) shows the constraints. The area marked by the constraints (see Figure 3.7 (b)) is the solution space. All valid solutions lie in that area. The objective function (3.23) is drawn in Figure 3.7 (b) for three different objective values (10, 20, 36). Obviously, they all have the same slope. All solutions that yield a certain objective value have to be in the solution space – otherwise they would break one or more constraints – and they have to touch the objective function for that objective value.

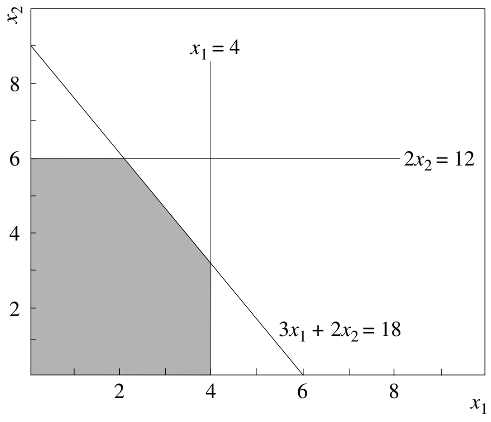
The goal is to find the objective function with the highest objective value that still touches the solution space and therefore contains at least one valid solution. This can be done graphically by shifting an objective function parallel towards the boundary of the solution space as indicated in Figure 3.7 (b). This solution – or these solutions as there can be multiple solutions that yield the same objective value – is the optimal solution. In the example shown previously, the optimal solution is $x_1 = 2$, $x_2 = 6$, which yields an objective value of 36 (which stands for a profit of \$36,000).

This graphical solution method works well with small problems of two and to some extent, three variables. We discuss solution techniques for much larger problems next.

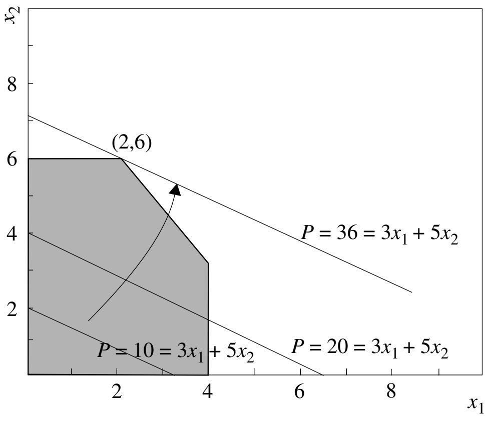
Simplex Several well-researched methods for solving LP problems exist. The oldest algorithm is the *simplex* algorithm presented in Dantzig (1951). It is the most commonly used method. There are different versions of the simplex algorithm. The original algorithm is most commonly found in textbooks as it is a straightforward algebraic procedure. However, there are advanced versions of the algorithm that can be implemented more efficiently. A widely used efficient version of the algorithm, the so-called revised simplex algorithm, can be found in Hillier and Lieberman (2001), Chapter 5.2.

There are several open source implementations of the simplex algorithm available. A very useful free open source tool is lp_solve which is available at http://lpsolve.sourceforge.net.

Alternatives There are various alternatives to the simplex algorithm. The *Khachiyan ellipsoid* method described in Khachiyan (1979), for example, has polynomial running



(a) Constraints



(b) Solution Space and Objective Functions

Figure 3.7 Graphical Solution of the Example

time, but is often found slow and inefficient in practice, see Korte and Vygen (2002). Also polynomial in time but faster in practice is the *interior point* method, see Karmarkar (1984).

3.3.3.2 Integer Programming Problems

Mixed integer programming (MIP) problems are much harder to solve than LP problems. For them, no general algorithm with polynomial time is known. A standard approach to solving MIPs is *branch & bound* with *LP relaxation*:

- An integer programming problem has only a finite number of feasible solutions owing to the fact that the variables can only take integer values. A full enumeration could be used to evaluate all feasible solutions and find the optimum one, but that would not be efficient as the number of feasible solutions is often extremely large. The branch & bound strategy is an enumeration method that tries to look at only a tiny faction of all feasible solutions while still finding the optimal solution. It consists of two steps, the branching and the bounding step. Both steps are executed many times, until the optimal solution is found.
- In each *branching* step, the set of feasible solutions is split subsequently into smaller and smaller subsets. This can be done in different ways. A standard approach is to pick one integer variable and force it to 0 in one subset, to 1 in the next, and so on. The algorithm has to keep track of all subsets, for example, by storing them in a tree-like data structure. All subsets are evaluated (one per bounding step) and are
  - either discarded,
  - split into further subsets (branching again) that then have to be evaluated,
  - or the best valid solution so far is found in the subset during the bounding phase.

  In the last case, that solution is stored. If later a better solution is found, the better solution replaces the old one. When all subsets have been evaluated, the best stored solution is the optimal solution of the whole optimisation problem. If no valid solution is found at all in that process, the whole problem is infeasible.
- In the *bounding* steps, the subsets are evaluated by solving a *relaxation* of the optimisation problem on that subset. The relaxation is chosen such that the relaxed problem can be solved relatively quickly. An (mixed) integer programming problem can always be relaxed towards an LP problem by making the variables continuous. This is called *LP relaxation*. LP relaxation increases the solution space because non-integer solutions become valid. The LP relaxation can be solved with simplex or the other methods discussed above. Looking at the optimal solution of the LP relaxation, two situations are possible:
  - The optimal solution of the relaxation might include only integer values for the variables. In that case, this solution is valid for the original integer programming problem. The objective value is compared with the best valid solution found so far and the better of the two solutions is retained.
  - Often, however, the optimal solution of the relaxation also includes non-integer values for one or more variables. In that case, the solution is valid for the LP relaxation but not for the original problem. Still, the objective value of the solution is a bound for the best valid solution for the original problem in this subset. A valid

integer solution can never be better than the best solution of the relaxation. This bound can be used to check whether this subset can be discarded or not:

- If the bound is not better than the best valid solution found so far, this subset can be safely discarded. It cannot contain an integer solution that could beat the currently best solution.
- If the bound is better than the best integer solution, this subset cannot be discarded and a branching step has to be performed on the current subset to split it into further subsets that are evaluated in later bounding steps.

When the branch & bound is finished, it returns the optimal solution. The running time is normally drastically smaller than a full enumeration, but that is not guaranteed and depends on the exact branching strategy and the order of bounding steps.

The branch & bound algorithm can be interrupted any time. The gap between the best bound of non-discarded subsets and the objective value of the currently best integer solution is called the *optimality gap*. The optimal solution of the problem cannot beat the currently best solution by more than this gap.

Besides branch & bound, the cutting plane method and the Lagrangian relaxation are other methods for solving MIP problems. For more information on MIPs we recommend Korte and Vygen (2002); Schrijver (2003); Wolsey and Nemhauser (1999).

There are commercial and free open source software packages available for solving MIPs. Ilog CPLEX (see http://www.ilog.com) is probably the most famous commercial package and lp_solve (see http://lpsolve.sourceforge.net) the most famous open source package. Both are easy to use as standalone products to solve an occasional problem but can also be integrated easily into other software.

While methods for finding the optimal solution for MIP problems are well researched, they can be computational and memory intensive for certain problem structures. In that case, the use of a MIP solver can still be useful. Typical MIP solution strategies such as branch & bound with LP relaxation as explained above give an upper bound on how much better the optimal solution can be, compared to the best solution found so far. These solution strategies can also be interrupted after some time while maintaining control over the quality of the solution, especially compared to meta-heuristic approaches such as simulated annealing, tabu search and genetic algorithms, see Rayward-Smith *et al*. (1996).

3.4 Summary and Conclusions

In this chapter, we presented some basic methods for analysing, predicting, and optimising the performance of networks. Queueing theory is the standard technique for stochastic queueing systems. It is employed in various ways for different types of networks and queues. It mainly provides results about the average properties of a queue or a network of queues.

Network calculus was originally developed for deterministic queueing systems and employed for the Intserv guaranteed service (see Section 6.2.2.4). Deterministic network calculus is conservative and based on worst-case assumption, which can lead to a waste of resources. Therefore, network calculus is currently being extended to stochastic queueing systems. In the third section of this chapter, optimisation models were discussed and methods to solve optimisation problems were presented.