Network Architectures and Algorithms

R. Srikant
ECE and CSL
University of Illinois
rsrikant@illinois.edu

Lei Ying
ECE
Iowa State University
leiying@iastate.edu

June 3, 2011

Contents

Chapter 1

Mathematics of Internet Architecture

1.1 Mathematical Background: Convex Optimization

In this section, we present some basic results from convex optimization which we will find useful in the rest of the chapter. Often, the results will be presented without proofs, but some concepts will be illustrated with figures to provide an intuitive feel for the results.

1.1.1 Convex Sets and Convex Functions

We first introduce the basic concepts from optimization theory, including the definitions of convex sets and convex functions.

Definition 1.1.1 (Convex Set) *A set $\mathcal{S} \subseteq \mathcal{R}^n$ is convex if $\alpha x + (1-\alpha)y \in \mathcal{S}$ whenever $x, y \in \mathcal{S}$ and $\alpha \in [0,1]$. Since $\alpha x + (1-\alpha)y$, for $\alpha \in [0,1]$, describes the line segment between x and y, a convex set can be pictorially depicted as in Figure 1.1: Given any two points $x, y \in \mathcal{S}$, the line segment between x and y lies entirely in $\mathcal{S}$.* □

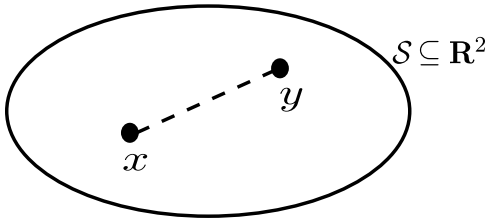

Figure 1.1: A convex set $\mathcal{S} \subseteq \mathcal{R}^2$

Definition 1.1.2 (Convex Hull) *The convex hull of set $\mathcal{S}$, denoted by $Co(\mathcal{S})$ is the smallest convex set that contains $\mathcal{S}$. See Figure 1.2 for an example.* □

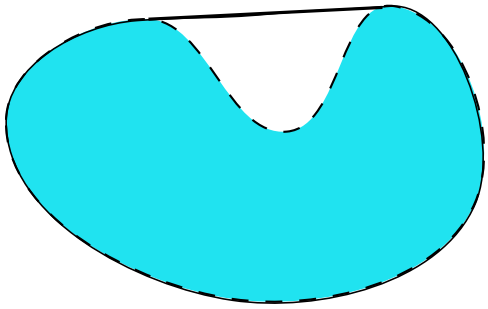

Figure 1.2: The solid line forms the boundary of the convex hull of the shaded set.

Definition 1.1.3 (Convex Function) *A function $f(x) : \mathcal{S} \subseteq \mathcal{R}^n \to \mathcal{R}$ is a convex function if $\mathcal{S}$ is a convex set and the following inequality holds for any $x, y \in \mathcal{S}$ and $\alpha \in [0, 1]$:*

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y).$$

$f(x)$ is strictly convex if the inequality above is strict for all $\alpha \in (0, 1)$ and $x \neq y$. Pictorially, $f(x)$ looks like a bowl as shown in Figure 1.3. □

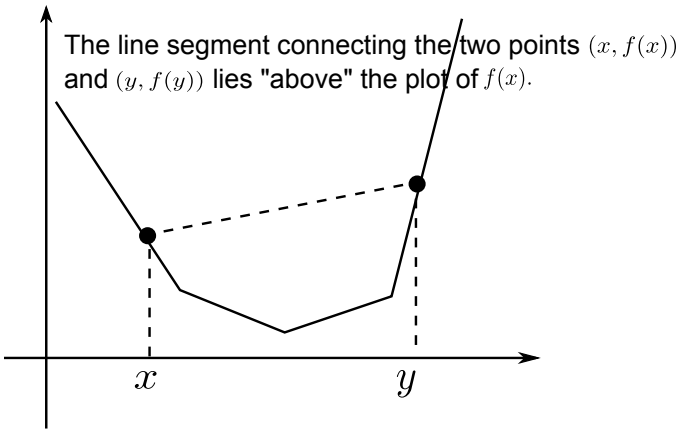

Figure 1.3: Pictorial description of a convex function in $\mathcal{R}^2$

Definition 1.1.4 (Concave Function) *A function $f(x) : \mathcal{S} \subseteq \mathcal{R}^n \to \mathcal{R}$ is a concave function (strictly concave) if $-f$ is a convex (strictly convex) function. Pictorially, $f(x)$ looks like an inverted bowl as shown in 1.4.* □

Definition 1.1.5 (Affine Function) *A function $f(x) : \mathcal{R}^n \to \mathcal{R}^m$ is an affine function if it is a sum of a linear function and a constant, i.e., there exist $\alpha, a \in \mathcal{R}$ such that*

$$f(x) = \alpha x + a.$$

□

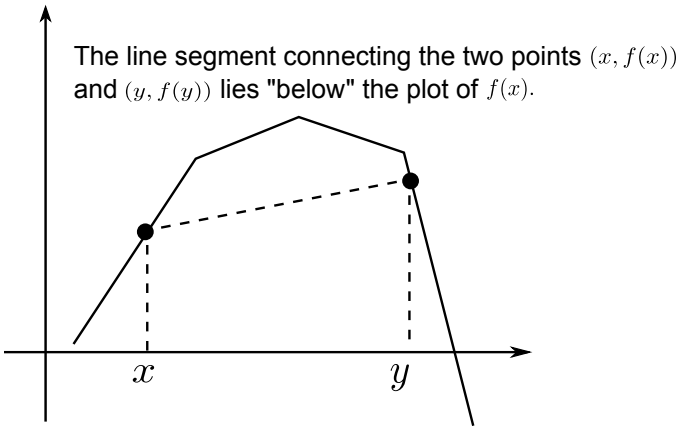

Figure 1.4: Pictorial description of a concave function in $\mathcal{R}^2$

The convexity of a function may be hard to verify from the definition given above. Therefore, next we present several conditions that can be used to verify the convexity of a function. The proofs are omitted here, and can be found in most textbooks on convex analysis or convex optimization.

Result 1.1.1 (First Order Condition I) *Let $f : \mathcal{S} \subset \mathcal{R} \to \mathcal{R}$ be a function defined over a convex set $\mathcal{S}$. If f is differentiable and the derivative $f'(x)$ is non-decreasing (increasing) in $\mathcal{S}$, then $f(x)$ is convex (strictly convex over $\mathcal{S}$).* □

Result 1.1.2 (First Order Condition II) *Let $f : \mathcal{S} \subset \mathcal{R}^n \to \mathcal{R}$ be a differentiable function defined over a convex set $\mathcal{S}$. Then f is a convex function if and only if*

$$f(y) \geq f(x) + \nabla f(x)(y - x) \ \forall x, y \in \mathcal{S}, \tag{1.1}$$

where

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x), \cdots, \frac{\partial f}{\partial x_n}(x) \right)$$

and x_i is the i^{th} component of vector x. Pictorially, if x is one-dimensional, this condition implies that the tangent of the function at any point lies below the function as shown in Figure 1.5.

$f(x)$ is strictly convex if the inequality above is strict for any $x \neq y$. □

Result 1.1.3 (Second Order Condition) *Let $f : \mathcal{S} \subset \mathcal{R}^n \to \mathcal{R}$ be a twice differentiable function defined over the convex set $\mathcal{S}$. Then, f is a convex (strictly convex) function if the Hessian matrix* $\mathbf{H}$ *with*

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}(x)$$

is positive semidefinite (positive definite). □

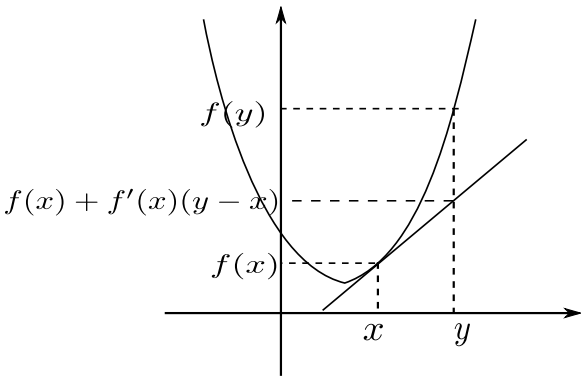

Figure 1.5: Pictorial description of inequality (1.1) in one-dimensional space

Result 1.1.4 (Strict Separation Theorem) *Let $\mathcal{S} \subset \mathcal{R}^n$ be a convex set and x be a point that is not contained in $\mathcal{S}$. Then there exists a vector $\beta \in \mathcal{R}^n$, $\beta \neq 0$, and constant $\delta > 0$ such that*

$$\sum_{i=1}^{n} \beta_i y_i \leq \sum_{i=1}^{n} \beta_i x_i - \delta$$

holds for any $y \in \mathcal{S}$. □

1.1.2 Convex Optimization

We first consider the following unconstrained optimization problem:

$$\max_{x \in \mathcal{S}} f(x), \tag{1.2}$$

and present some important results without proofs.

Definition 1.1.6 (Local Maximizer and Global Maximizer) *For any function $f(x)$ over $\mathcal{S} \subseteq \mathcal{R}^n$, x^* is said to be a* local *maximizer or* local *optimal point if there exists an $\epsilon > 0$ such that*

$$f(x^* + \delta x) \leq f(x^*)$$

for any $\|\delta x\| \leq \epsilon$, where $\|\cdot\|$ can be any norm. x^* is said to be a global *maximizer or* global *optimal point if*

$$f(x) \leq f(x^*)$$

for any $x \in \mathcal{S}$. When not specified, maximizer refers to global maximizer in this book. □

Result 1.1.5 *If $f(x)$ is a continuous function over a compact set $\mathcal{S}$ (i.e., $\mathcal{S}$ is closed and bounded if $\mathcal{S} \subseteq \mathcal{R}^n$), then $f(x)$ achieves a maximum over this set, i.e., $\max_{x \in \mathcal{S}} f(x)$ exists.* □

Result 1.1.6 *If $f(x)$ is differentiable, then any local maximizer x^* in the interior of $\mathcal{S} \subseteq \mathcal{R}^n$ satisfies*

$$\nabla f(x^*) = 0. \tag{1.3}$$

If $f(x)$ is a concave function over $\mathcal{S}$, condition (1.3) is also sufficient for x^* to be a local maximizer. □

Result 1.1.7 *If $f(x)$ is concave, then a local maximizer is also a global maximizer. In general, multiple global maximizers may exist. If $f(x)$ is strictly concave, then the global maximizer x^* is unique.* □

Result 1.1.8 *Results 1.1.6 and 1.1.7 hold for convex functions if the* max *in the optimization problem (1.2) is replaced by* min, *and maximizer is replaced by minimizer in Results 1.1.6 and 1.1.7.* □

Result 1.1.9 *If $f(x)$ is a differentiable function over set $\mathcal{S}$ and x^* is a maximizer of the function, then*

$$\nabla f(x^*)\delta x \leq 0$$

for any feasible direction δx, i.e., for any δx such that $x^* + \delta x \in \mathcal{S}$.

Further if $f(x)$ is a concave function, then x^* is a maximizer if and only if

$$\nabla f(x^*)\delta x \leq 0$$

for any δx such that $x + \delta x \in \mathcal{S}$. □

Next, we consider an optimization problem with equality and inequality constraints as follows:

$$\max_{x\in\mathcal{S}} f(x) \tag{1.4}$$

$$\text{subject to} \quad h_i(x) \leq 0, i = 1, 2, ..., I \tag{1.5}$$

$$g_j(x) = 0, j = 1, 2, ..., J. \tag{1.6}$$

A vector x is said to be *feasible* if $x \in \mathcal{S}$, $h_i(x) \leq 0$ for all i, and $g_j(x) = 0$ for all j. While (1.5) and (1.6) are inequality and equality constraints, respectively, the set $\mathcal{S}$ in the above problem captures any other constraints that are not in equality or inequality form.

A key concept that we will exploit later in the chapter is called Lagrangian duality. Duality refers to the fact that the above maximization problem, also called the *primal* problem, is closely related to an associated problem called the *dual* problem. Given the constrained optimization problem in (1.4)-(1.6), the *Lagrangian* of this optimization problem is defined to be

$$L(x, \lambda, \mu) = f(x) - \sum_{i=1}^{I} \lambda_i h_i(x) + \sum_{j=1}^{J} \mu_j g_j(x), \qquad \lambda_i \geq 0 \ \forall i.$$

The constants $\lambda_i \geq 0$ and μ_j are called Lagrange multipliers. The *Lagrangian dual function* is defined to be

$$D(\lambda, \mu) = \sup_{x\in\mathcal{S}} L(x, \lambda, \mu).$$

Let f^* be the maximum of the optimization problem (1.4), i.e., $f^* = \max_{x\in\mathcal{S}} f(x)$. Then, we have the following theorem.

Theorem 1.1.1 *$D(\lambda, \mu)$ is a convex function and $D(\lambda, \mu) \leq f^*$.*

Proof The convexity comes from a known fact that the pointwise supremum of affine functions is convex (see Figure 1.6). To prove the bound, note that $h_i(x) \leq 0$ and $g_j(x) = 0$ for any feasible x, so the following inequality holds for any feasible x,

$$L(x, \lambda, \mu) \geq f(x).$$

This inequality further implies that

$$\sup_{\substack{x \in \mathcal{S} \\ h(x) \leq 0 \\ g(x) = 0}} L(x, \lambda, \mu) \geq \sup_{\substack{x \in \mathcal{S} \\ h(x) \leq 0 \\ g(x) = 0}} f(x) = f^*.$$

Since removing some constraints of a maximization problem can only result in a larger maximum value, we obtain

$$\sup_{x \in \mathcal{S}} L(x, \lambda, \mu) \quad \geq \quad \sup_{\substack{x \in \mathcal{S} \\ h(x) \leq 0 \\ g(x) = 0}} L(x, \lambda, \mu).$$

Therefore, we conclude that

$$D(\lambda, \mu) = \sup_{x \in \mathcal{S}} L(x, \lambda, \mu) \geq f^*.$$

□

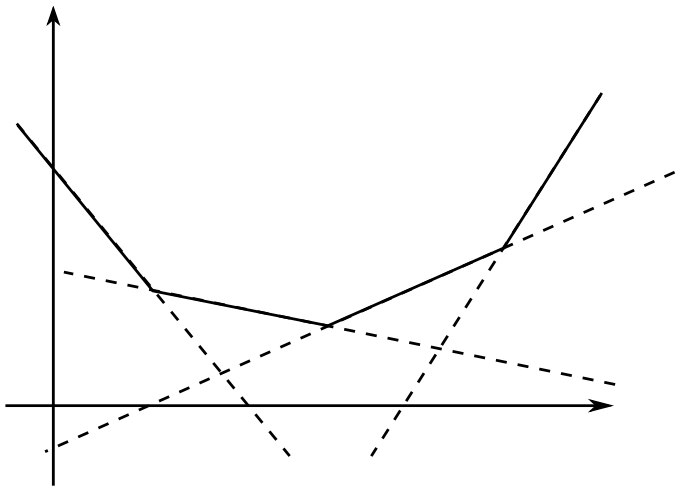

Figure 1.6: The solid-line is the pointwise supremum of the four dashed-lines, and is convex.

Theorem 1.1.1 states that the dual function is an upper bound on the maximum of the optimization problem (1.4)-(1.6). We can optimize over λ and μ to obtain the best upper bound, which yields the following minimization problem, called the *Lagrange dual problem*:

$$\inf_{\lambda \geq 0, \mu} D(\lambda, \mu).$$

Let d^* be the minimum of the dual problem, i.e., $d^* = \inf_{\lambda \geq 0, \mu} D(\lambda, \mu)$. The difference between d^* and f^* is called the duality gap. For some problems, the duality gap is zero. We say *strong duality* holds if $d^* = f^*$. If strong duality holds, then one can solve either the primal problem or the dual problem to obtain f^*. This is often helpful since sometimes one of the problems is easier to solve than the other. A simple yet frequently used condition to check strong duality is *Slater's condition,* which is given below.

Theorem 1.1.2 (Slater's condition) *Consider the constrained optimization problem defined by (1.4)-(1.6). Strong duality holds if the following conditions are true:*

- *$f(x)$ is a concave function and $h_i(x)$ are convex functions.*
- *$g_j(x)$ are affine functions.*
- *There exists an x that belongs to the relative interior[1] of $\mathcal{S}$ such that $h_i(x) < 0$ for all i and $g_j(x) = 0$ for all j.*

□

As mentioned earlier, when strong duality holds, we have a choice of solving the original optimization in one of two ways: either solve the primal problem directly or solve the dual problem. Later in this chapter, we will see that resource allocation problems in communication networks can be posed as convex optimization problems, and we can use either the primal or the dual formulations to solve the resource allocation problem. We now present a result which can be used to solve convex optimization problems.

Theorem 1.1.3 (Karush-Kuhn-Tucker (KKT) Conditions) *Consider the constrained optimization problem defined in (1.4)-(1.6), where f is a concave function, h_i $(i = 1, \ldots, I)$ are convex functions and g_j $(j = 1, \ldots, J)$ are affine functions. Let x^* be a feasible point, i.e., a point that satisfies all the constraints. Suppose there exist constants $\lambda_i^* \geq 0$ and μ_j^* such that*

$$\frac{\partial f}{\partial x_k}(x^*) - \sum_i \lambda_i^* \frac{\partial h_i}{\partial x_k}(x^*) + \sum_j \mu_j^* \frac{\partial g_j}{\partial x_k}(x^*) = 0, \quad \forall k, \tag{1.7}$$

$$\lambda_i^* h_i(x^*) = 0, \quad \forall i, \tag{1.8}$$

then x^* is a global maximizer of the constrained optimization problem, (λ^*, μ^*) is a global minimizer of the Lagrange dual problem, and strong duality holds. If f is strictly concave, then x^* is also the unique global maximizer. □

The KKT conditions (1.7)-(1.8) can be interpreted as follows. Consider the Lagrangian

$$L(x, \lambda, \mu) = f(x) - \sum_i \lambda_i h_i(x) + \sum_j \mu_j g_j(x).$$

[1]For convex set $\mathcal{S}$, a relative interior is a point x such that for any $y \in \mathcal{S}$ there exist $z \in \mathcal{S}$ and $0 < \lambda < 1$ such that $x = \lambda y + (1 - \lambda) z$.

Condition (1.7) is the first-order necessary condition for the maximization problem $\max_{x\in\mathcal{S}} L(x, \lambda^*, \mu^*)$. When strong duality holds, we have

$$f(x^*) = f(x^*) - \sum_i \lambda_i^* h_i(x^*) + \sum_j \mu_j^* g_j(x^*),$$

which results in condition (1.8) since $g_j(x^*) = 0\ \forall j$, and $\lambda_i^* \geq 0$ and $h_i(x^*) \leq 0\ \forall i$. We remark that condition (1.8) is called complementary slackness.

1.2 Resource Allocation as Utility Maximization

The Internet is a shared resource, shared by many millions of users, who are connected by a huge network consisting of many, many routers and links. The capacity of the links must be split in some *fair* manner among the users. To appreciate the difficulty in defining what fairness means, let us consider an every day example. Suppose that one has a loaf of bread which has to be divided among three people. Almost everyone will agree that the fair allocation is to divide the loaf into three equal parts and give one piece to each person. While this seems obvious, consider a slight variant of the situation where one of the people is a two-year old child and the other two are football players. Then, an equal division does not seem appropriate: the child cannot possibly consume the third allocated to her, so a different division based on their needs may appear to be more appropriate. The situation gets more complicated when there is more than one resource to be divided among the three people. Suppose that there are two loafs of bread, one wheat and one rye, then a fair division has to take into account the preferences of the individuals for the different types of bread. Economists solve such problems by associating a so-called utility function with each individual, and finding an allocation that maximizes the net utility of the individuals. We now formally describe and model the resource allocation problem in the Internet.

Suppose we have a network with a set of traffic sources $\mathcal{S}$ and a set of links $\mathcal{L}$. Each link $l \in \mathcal{L}$ has a finite fixed capacity c_l. Each source in $\mathcal{S}$ is associated with a fixed route $r \subset \mathcal{L}$ along which it transmits at some rate x_r to a fixed destination. In our model, a route is simply a collection of links connecting a source to its destination. In fact, the order of the links in the route is irrelevant for our mathematical model. Note that we can use the index r to indicate both a route and the source that sends traffic along that route and we will follow this notation. Also since multiple sources could use the same set of links as their routes, there could be multiple indices r which denote the same subset of $\mathcal{L}$. The utility that the source obtains from transmitting data on route r at rate x_r is denoted by $U_r(x_r)$. We assume that the utility function is continuously differentiable, non-decreasing and strictly concave. The concavity assumption follows from the diminishing returns idea—a person downloading a file would appreciate the effect of a rate increase from 1 kbps to 100 kbps much more than an increase from 1 Mbps to 1.1 Mbps although the increase is the same in both cases.

The goal of resource allocation is to solve the following optimization problem, called Network Utility Maximization (NUM):

$$\max_{x_r} \sum_{r\in\mathcal{S}} U_r(x_r) \tag{1.9}$$

subject to the constraints

$$\sum_{r:l\in r} x_r \leq c_l, \ \forall l \in \mathcal{L}, \tag{1.10}$$

$$x_r \geq 0, \ \forall r \in \mathcal{S}. \tag{1.11}$$

The above inequalities state that the capacity constraints of the links cannot be violated and that each source must be allocated a non-negative rate of transmission. The utility maximization problem has a unique solution since *a strictly concave function has a unique maximizer over a closed and bounded set*. In addition, the constraint set for the utility maximization problem is convex which allows us to use the method of Lagrange multipliers and the Karush-Kuhn-Tucker (KKT) theorem to solve the optimal solution. We consider an example of such a maximization problem in a small network and show how one can solve the problem using the above method of Lagrange multipliers.

Example 1 *Consider the network in Figure 1.7 in which three sources compete for resources in the core of the network. Links* L_1, L_3 *and* L_5 *have a capacity of* 2 *units per second, while links* L_2 *and* L_4 *have capacity* 1 *unit per second. There are three flows and denote their data rates by* x_0, x_1 *and* x_2.

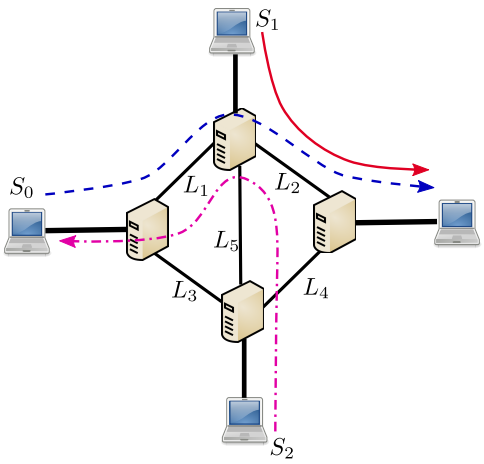

Figure 1.7: Example illustrating network resource allocation. We assume that links L_1, L_3 and L_5 have capacity 2, while L_2 and L_4 have capacity 1. The access links of the sources are assumed to have infinite capacity. There are three flows in the system.

In our problem, links L_3 *and* L_4 *are not used, while* L_5 *does not constrain source* S_2. *Assuming log utility functions, the resource allocation problem is given by*

$$\max_x \sum_{r=0}^{2} \log x_r \tag{1.12}$$

with constraints

$$\begin{aligned} x_0 + x_1 &\leq 1, \\ x_0 + x_2 &\leq 2, \\ x &\geq 0, \end{aligned}$$

where x is the vector consisting of x_0, x_1 and x_2. Now, since $\log x \to -\infty$ as $x \to 0$, the optimal resource allocation will not yield a zero rate for any source, even if we remove the non-negativity constraints. So the last constraint above is not active.

We define p_1 and p_2 to be the Lagrange multipliers *corresponding to the capacity constraints on links L_1 and L_2, respectively, and let p denote the vector of Lagrange multipliers. Then, the* Lagrangian *is given by*

$$L(x, p) = \log x_0 + \log x_1 + \log x_2 - p_1(x_0 + x_1) - p_2(x_0 + x_2).$$

Setting $\frac{\partial L}{\partial x_r} = 0$ for each r gives

$$x_0 = \frac{1}{p_1 + p_2}, \quad x_1 = \frac{1}{p_1}, \quad x_2 = \frac{1}{p_2}. \tag{1.13}$$

Letting $x_0 + x_1 = 2$ and $x_0 + x_2 = 1$ (since we can always increase x_1 or x_2 until this is true) yields

$$p_1 = \frac{\sqrt{3}}{\sqrt{3}+1} = 0.634, \qquad p_2 = \sqrt{3} = 1.732.$$

Note that (1.8) is automatically satisfied since $x_0 + x_1 = 2$ and $x_0 + x_2 = 1$, and (1.7) is satisfied due to equalities (1.13). Therefore, the values of the Lagrange multipliers actually are the minimizers of the dual function. Hence, we have the final optimal allocation

$$x_0^* = \frac{\sqrt{3}+1}{3+2\sqrt{3}} = 0.422, \quad x_1^* = \frac{\sqrt{3}+1}{\sqrt{3}} = 1.577, \quad x_2^* = \frac{1}{\sqrt{3}} = 0.577.$$

A few facts are noteworthy in this simple network scenario:

- *Note that $x_1 = 1/p_1$, and it does not depend on p_2 explicitly. Similarly, x_2 does not depend on p_1 explicitly. In general, we will see later that the optimal transmission rate for source r is only determined by the Lagrange multipliers on its route. We will also see that this feature is extremely useful in designing decentralized algorithms to reach the optimal solution.*

- *The value of x_r is inversely proportional to the sum of the Lagrange multipliers on its route. We will see later that, in general, x_r is a decreasing function of the Lagrange multipliers. Thus, the Lagrange multiplier associated with a link can be thought of the price for using that link and the price of a route can be thought of as the sum of the prices of its links. If the price of a route increases, then the transmission rate of a source using that route decreases.*

□

In the above example, it was easy to solve the Lagrangian formulation of the problem since the network was small. In the Internet which consists of thousands of links and possibly millions of users, such an approach is not possible. In the next section, we will see that there are distributed solutions to the optimization problem which are easy to implement in the Internet.

1.2.1 Notions of Fairness

In our discussion of the network utilization maximization, we have associated a utility function with each user. The utility function can be viewed as a measure of satisfaction of the user when it gets a certain data rate from the network. A different point of view is that a utility function is assigned to each user in the network by a service provider with the goal of achieving a certain type of resource allocation. For example, suppose $U(x_r) = \log x_r$, for all users r. It is a well-known property of concave functions that

$$\nabla f(x^*)(x - x^*) \leq 0, \tag{1.14}$$

where x^* is the maximizer of $f(x)$. So the optimal rates which solve the network utility maximization problem, $\{x_r^*\}$, satisfy

$$\sum_r \frac{x_r - x_r^*}{x_r^*} \leq 0,$$

where $\{x_r\}$ is any other set of feasible rates. For log utility functions, this property states that, under any other allocation, the sum of proportional changes in the users' utilities will be non-positive. Thus, if some User A's rate increases, then there will be at least one other user whose rate will decrease and further, the proportion by which it decreases will be larger than the proportion by which the rate increases for User A. Therefore, such an allocation is called *proportionally fair*. If the utilities are chosen such that $U_r(x_r) = w_r \log x_r$, where $w_r \geq 0$ is some weight, then the resulting allocation is said to be *weighted proportionally fair*.

Another widely used fairness criterion in communication networks is called *max-min* fairness. An allocation $\{x_r^*\}$ is called max-min fair if it satisfies the following property: if there is any other allocation $\{x_r\}$ such a user s's rate increases, i.e., $x_s > x_s^*$, then there has to be another user u with the property

$$x_u < x_u^* \qquad \text{and} \qquad x_u^* \leq x_s^*.$$

In other words, if we attempt to increase the rate for one user, then the rate for a less-fortunate user will suffer. The definition of max-min fairness implies that

$$\min_r x_r^* \geq \min_r x_r,$$

for any other allocation $\{x_r\}$. To see why this is true, suppose that there exists an allocation such that

$$\min_r x_r^* < \min_r x_r. \tag{1.15}$$

This implies that, for any s such that $\min_r x_r^* = x_s^*$, the following holds: $x_s^* < x_s$. Otherwise, our assumption (1.15) cannot hold. However, this implies that if we switch the allocation from $\{x_r^*\}$ to $\{x_r\}$, then we have increased the allocation for s without affecting a less-fortunate user (since there is no less-fortunate user than s under $\{x_r^*\}$). Thus, the max-min fair resource allocation attempts to first satisfy the needs of the user who gets the least amount of resources from the network. In fact, this property continues to hold if we remove all the users whose rates are the smallest under max-min fair allocation, reduce the link capacities by the amounts used by these users and consider the resource allocation for the rest of the users. The same argument as above applies. Thus, max-min is a very egalitarian notion of fairness.

Yet another form of fairness that has been discussed in the literature is called *minimum potential delay* fairness. Under this form of fairness, user r is associated with the utility function $-1/x_r$. The goal of maximizing the sum of the user utilities is equivalent to minimizing $\sum_r 1/x_r$. The term $1/x_r$ can be interpreted as follows: suppose user r needs to transfer a file of unit size. Then, $1/x_r$ is the delay in associated with completing this file transfer since the delay is simply the file size divided by the rate allocated to user r. Hence, name *minimum potential delay* fairness.

All of the above notions of fairness can be captured by using utility functions of the form

$$U_r(x_r) = \frac{x_r^{1-\alpha}}{1-\alpha}, \tag{1.16}$$

for some $\alpha > 0$. Resource allocation using the above utility function is called α-*fair*. Different values of α yield different ideas of fairness. First consider $\alpha = 2$. This immediately yields minimum potential delay fairness. Next, consider the case $\alpha = 1$. Clearly, the utility function is not well-defined at this point. But it is instructive to consider the limit $\alpha \to 1$. Notice that maximizing the sum of $\frac{x_r^{1-\alpha}}{1-\alpha}$ yields the same optimum as maximizing the sum of

$$\frac{x_r^{1-\alpha} - 1}{1-\alpha}.$$

Now, by applying L'Hospital's rule, we get

$$\lim_{\alpha \to 1} \frac{x_r^{1-\alpha} - 1}{1-\alpha} = \log x_r,$$

thus yielding proportional fairness in the limit as $\alpha \to 1$.

Next, we argue that the limit $\alpha \to \infty$ gives max-min fairness. Let $x_r^*(\alpha)$ be the α-fair allocation. Assume that $x_r^*(\alpha) \to x_r^*$ as $\alpha \to \infty$ and $x_1^* < x_2^* < \ldots < x_n^*$. Let ϵ be the minimum difference of $\{x_r^*\}$, i.e., $\epsilon = \min_r(x_{r+1}^* - x_r^*)$. Then when α is sufficiently large, we have $|x_r^*(\alpha) - x_r^*| \leq \epsilon/4$, which implies that $x_1^*(\alpha) < x_2^*(\alpha) < \ldots < x_n^*(\alpha)$.

Now by the property of concave functions mentioned earlier (inequality (1.14)),

$$\sum_r \frac{x_r - x_r^*(\alpha)}{x_r^{*\alpha}(\alpha)} \leq 0.$$

Considering an arbitrary flow s, the above expression can be rewritten as

$$\sum_{r=1}^{s} (x_r - x_r^*(\alpha)) \frac{x_s^{*\alpha}(\alpha)}{x_r^{*\alpha}(\alpha)} + (x_s - x_s^*(\alpha)) + \sum_{i=s+1}^{n} (x_i - x_i^*(\alpha)) \frac{x_s^{*\alpha}(\alpha)}{x_i^{*\alpha}(\alpha)} \leq 0.$$

Since $|x_r^*(\alpha) - x_r^*| \leq \epsilon/4$, we further have

$$\sum_{r=1}^{s} (x_r - x_r^*(\alpha)) \frac{x_s^{*\alpha}(\alpha)}{x_r^{*\alpha}(\alpha)} + (x_s - x_s^*(\alpha)) - \sum_{i=s+1}^{n} |x_i - x_i^*(\alpha)| \frac{(x_s^* + \epsilon/4)^\alpha}{(x_i^* - \epsilon/4)^\alpha} \leq 0.$$

Note that $x_i^* - \epsilon/4 - (x_s^* + \epsilon/4) \geq \epsilon/2$ for any $i > s$, so by increasing α, the third term in the above expression will become negligible. Thus, if $x_s > x_s^*(\alpha)$, then the allocation for at least one user whose rate satisfies $x_r^*(\alpha) < x_s^*(\alpha)$ will decrease. The argument can be made rigorous and extended to the case $x_r^* = x_s^*$ for some r and s. Therefore as $\alpha \to \infty$, the α-fair allocation approaches max-min fairness.

1.3 Mathematical Background: Stability of Dynamical Systems

Consider a dynamical system defined by the following differential equation

$$\dot{x} = f(x), \qquad f : \mathcal{R}^n \to \mathcal{R}^n, \tag{1.17}$$

where $\dot{x}$ is the derivative of x with respect to the time t. The time variable t has been omitted in most of places when no confusion is caused. Assume that $x(0)$ is given. Throughout we will assume that f is a continuous function and that it also satisfies other appropriate conditions to ensure that the differential equation has a unique solution $x(t)$, for $t \geq 0$.

A point $x_e \in \mathcal{R}^n$ is said to be the equilibrium point of the dynamical system if $f(x_e) = 0$. We assume that $x_e = 0$ is the *unique* equilibrium point of this dynamical system.

Definition 1.3.1 (Globally, asymptotically stable) *$x_e = 0$ is said to be a globally asymptotically stable equilibrium point if*

$$\lim_{t\to\infty} x(t) = 0$$

for any $x(0) \in \mathcal{R}^n$.

We first introduce the Lyapunov boundedness theorem.

Theorem 1.3.1 (Lyapunov boundedness theorem) *Let $V : \mathcal{R}^n \to \mathcal{R}$ be a differentiable function with the following property:*

$$V(x) \to \infty \text{ as } \|x\| \to \infty. \tag{1.18}$$

Denote by $\dot{V}(x)$ the derivative of $V(x)$ with respect to t, i.e.,

$$\dot{V}(x) = \nabla V(x)\dot{x} = \nabla V(x) f(x).$$

If $\dot{V}(x) \leq 0$ for all x, then there exists a constant $B > 0$ such that $\|x(t)\| \leq B$ for all t.

Proof At any time T, we have

$$V(x(T)) = V(x(0)) + \int_0^T \dot{V}(x(t))\, dt \leq V(x(0)).$$

Note that condition (1.18) implies that $\{x : V(x) \leq c\}$ is a bounded set for any c. Letting $c = V(x(0))$, the theorem follows.

□

Theorem 1.3.2 (Lyapunov global asymptotic stability theorem) *If in addition to the conditions in the previous theorem, we assume that $V(x)$ is continuously differentiable and also satisfies the following conditions:*

(1) $V(x) \geq 0$ $\forall x$ and $V(x) = 0$ if and only if $x = 0$.

(2) $\dot{V}(x) < 0$ for any $x \neq 0$ and $\dot{V}(0) = 0$.

Then, the equilibrium point $x_e = 0$ is globally, asymptotically stable.

Proof We prove this theorem by contradiction. Suppose $x(t)$ does not converge to the equilibrium point 0 as $t \to \infty$.

Note that $V(x(t))$ is non-increasing because its derivative with respect to t is non-positive ($\dot{V}(x) \leq 0$) for any x. Since $V(x(t))$ decreases as a function of t and is lower bounded (since $V(x) \geq 0 \; \forall x$), it converges as $t \to \infty$. Suppose that $V(x(t))$ converges to, say, $\epsilon > 0$. Define the set

$$\mathcal{C} \triangleq \{x : \epsilon \leq V(x) \leq V(x(0))\}.$$

The set $\mathcal{C}$ is bounded since $V(x) \to \infty$ as $||x|| \to \infty$ and it is closed since $V(x)$ is a continuous function of x. Thus, $\mathcal{C}$ is a compact set.

Let

$$-a = \sup_{x \in \mathcal{C}} \dot{V}(x)$$

where $a > 0$ is finite because $\dot{V}(x)$ is continuous in x and $\mathcal{C}$ is a compact set. Now we write $V(x(t))$ as

$$\begin{aligned} V(x(t)) &= V(x(0)) + \int_0^t \dot{V}(x(s))\, ds \\ &\leq V(x(0)) - at, \end{aligned}$$

which implies that

$$V(x(t)) = 0, \quad \forall \; t \geq \frac{V(x(0))}{a},$$

and

$$x(t) = 0, \quad \forall \; t \geq \frac{V(x(0))}{a}.$$

This contradicts with the assumption that $x(t)$ does not converge to 0.

□

The Lyapunov global asymptotic stability theorem requires that $\dot{V}(x) \neq 0$ for any $x \neq 0$. In the case $\dot{V}(x) = 0$ for some $x \neq 0$, global asymptotic stability can be studied using Lasalle's invariance principle. The proof of the theorem is omitted in this book

Theorem 1.3.3 (Lasalle's invariance principle) *Replace condition (2) of the previous theorem by*

$$\dot{V}(x) \leq 0 \qquad \forall x,$$

and suppose that the only trajectory $x(t)$ that satisfies

$$\dot{x}(t) = f(x(t)) \quad \text{and} \quad \dot{V}(x(t)) = 0, \quad \forall t$$

is $x(t) = 0 \; \forall t$. Then $x = 0$ is globally, asymptotically stable. □

1.4 Distributed Algorithms: Primal Solution

In the previous section, we formulated an optimization problem, the solution of which provided fair resource allocation. However, the technique used to solve the optimization problem in Example 1 assumed that we had complete knowledge of the topology and routes. Clearly this is infeasible in a giant network such as the Internet. In this section and the next, we will study distributed algorithms which only require limited information exchange among the sources and the network for implementation.

The approach in this section is called the primal solution. We first relax the capacity constraints: instead of requiring that the total arrival rate at each link is less than the capacity, we assume that there is a cost for sending data at a certain rate over a link. Then, instead of having these resource constraints in the optimization problem, we subtract the cost from the total utility of the sources in the network as follows:

$$W(x) = \sum_{r \in \mathcal{S}} U_r(x_r) - \sum_{l \in \mathcal{L}} B_l \left(\sum_{s:l \in s} x_s \right), \tag{1.19}$$

where x is the vector of rates of all sources and $B_l(.)$ is the cost of sending data on link l: it can be interpreted as either a "barrier" function associated with link l which increases to infinity when the arrival rate on link l approaches the link capacity c_l or a "penalty" function which penalizes the arrival rate for exceeding the link capacity. By appropriate choice of the function B_l, one can solve the exact utility optimization problem posed in the previous section; for example, choose $B_l(y)$ to be zero if $y \leq c_l$ and equal to ∞ if $y > c_l$. However, such a solution may not be desirable or required. For example, the design principle may be such that one requires the delays on all links to be small. While it is not apparent in the deterministic formulation here, later in the book we will see that even when the arrival rate on a link is less than its capacity, due to randomness in the arrival process, packets in the network will experience delay or packet loss. The function $B_l(.)$ may thus be used to represent average delay, packet loss rate, etc. Thus, $W(x)$ represents a tradeoff: large values of x_r increase utility, but result in packets incurring excessive delays or other impairments at the link.

We first assume that B_l is a convex function so that the function (1.19) is a strictly concave function.. Further, assume that B_l is continuously differentiable. Then, we can equivalently require that

$$B_l \left(\sum_{s:l \in s} x_s \right) = \int_0^{\sum_{s:l \in s} x_s} f_l(y) dy, \tag{1.20}$$

where $f_l(\cdot)$ is an increasing, continuous function. We call $f_l(y)$ the *congestion price function*, or simply the price function, associated with link l, since it associates a price with the level of congestion on the link. It is straightforward to see that B_l defined in the above fashion is convex, since integrating an increasing function results in a convex function (see Result 1.1.1).

We will assume that U_r and f_l are such that the maximization of (1.19) results in a solution with $x_r > 0 \ \forall r \in \mathcal{S}$. Now, the condition that must be satisfied by the maximizer of (1.19) is obtained by differentiation and is given by

$$U_r'(x_r) - \sum_{l:l \in r} f_l \left(\sum_{s:l \in s} x_s \right) = 0, \qquad r \in \mathcal{S}. \tag{1.21}$$

We now require a distributed algorithm that would drive x towards the solution of (1.21). A natural candidate for such an algorithm is the so-called gradient ascent algorithm from optimization theory.

The idea here is that if we want to maximize a function of the from $g(x)$, then we progressively change x so that $g(x(t+\delta)) > g(x(t))$. We do this by finding the direction in which a change in x produces the greatest increase in $g(x)$. This direction is given by the gradient of $g(x)$ with regard to x. In one dimension, we merely choose the update algorithm for x as

$$x(t+\delta) = x(t) + k(t)\frac{dg(x)}{dx}\delta,$$

where $k(t)$ is a scaling parameter which controls the amount of change in the direction of the gradient, or letting $\delta \to 0$

$$\dot{x} = k(t)\frac{dg(x)}{dx}. \tag{1.22}$$

Let us try to design a similar algorithm for the network utility maximization problem. Consider the algorithm

$$\dot{x}_r = k_r(x_r)\left(U_r'(x_r) - \sum_{l:l\in r} f_l\left(\sum_{s:l\in s} x_s\right)\right). \tag{1.23}$$

We have obtained the above by differentiating (1.19) with respect to x_r to find the direction of ascent, and used it along with a scaling function $k_r(\cdot)$ to construct an algorithm of the form shown in (1.22). The scaling function $k_r(\cdot)$ must be chosen such that the equilibrium of the differential equation is the same as the one obtained from (1.21). Thus, the equilibrium point of the differential equation is the same as the solution to the resource allocation problem. The controller is called a *primal algorithm* since it arises from the primal formulation of the utility maximization problem. Note that the primal algorithm has many intuitive properties that one would expect from a resource allocation/congestion control algorithm. When the route price $q_r = \sum_{l:l\in r} f_l(\sum_{s:l\in s} x_s)$ is large, then the congestion controller decreases its transmission rate. Further, if x_r is large, then $U'(x_r)$ is small (since $U_r(x_r)$ is concave) and thus the rate of increase is small as one would expect from a resource allocation algorithm which attempts to maximize the sum of the user utilities.

We must now answer two questions regarding the performance of the primal congestion control algorithm:

- What information is required at each source in order to implement the algorithm?
- Does the algorithm actually converge to the desired equilibrium point?

Below we consider the answer to the first question and develop a framework for answering the second. The precise answer to the convergence question will be presented in the next subsection.

The first question is easily answered by studying (1.23). It is clear that all that the source r needs to know in order to reach the optimal solution is the sum of the prices of each link on its route. How would the source be appraised of the link prices? The answer is to use a feedback mechanism—each packet generated by the source collects the price of each link that it traverses. When the destination receives the packet, it sends this price information in a small packet (called the acknowledgment packet or *ack* packet) that it sends back to the source.

To visualize this feedback control system, we introduce a matrix R which is called the routing matrix of the network. The (l, r) element of this matrix is given by

$$R_{lr} = \begin{cases} 1 & \text{if route } r \text{ uses link } l \\ 0 & \text{else} \end{cases}$$

Let us define

$$y_l = \sum_{s:l \in s} x_s, \tag{1.24}$$

which is the load on link l. Using the elements of the routing matrix, y_l can also be written as

$$y_l = \sum_s R_{ls} x_s.$$

Letting y be the vector of all y_l ($l \in \mathcal{L}$), we have

$$y = Rx. \tag{1.25}$$

Let $p_l(t)$ denote the price of link l at time t, i.e.,

$$\begin{aligned} p_l(t) &= f_l\left(\sum_{s:l \in s} x_s(t)\right) \\ &= f_l(y_l(t)). \end{aligned} \tag{1.26}$$

Then the price of a route is just the sum of link prices p_l of all the links in the route. So we define the price of route r to be

$$q_r = \sum_{l:l \in r} p_l. \tag{1.27}$$

Also let p be the vector of all link prices and q be the vector of all route prices. We thus have

$$q = R^T p \tag{1.28}$$

The relationships derived above can be made clear using the block diagram in Figure 1.8.

We show that the primal controller of (1.23) is globally asymptotically stable by using the Lyapunov function idea described in Section 1.3. Recall that $W(x)$ is a strictly concave function. Let $\hat{x}$ be its unique maximizer. Then, $W(\hat{x}) - W(x)$ is non-negative and is equal to zero only at $x = \hat{x}$. Thus, $W(\hat{x}) - W(x)$ is a natural candidate Lyapunov function for the system (1.23). We use this Lyapunov function in the following theorem.

Theorem 1.4.1 *Consider a network in which all sources follow the primal control algorithm (1.23). Assume that the functions $U_r(\cdot)$, $k_r(\cdot)$ and $f_l(\cdot)$ are such that $V(x) = W(\hat{x}) - W(x)$ is such that $V(x) \to \infty$ as $||x|| \to \infty$, $\hat{x}_i > 0$ for all i, and $W(x)$ is as defined in (1.19). Then, the controller in (1.23) is globally asymptotically stable and the equilibrium value maximizes (1.19).*

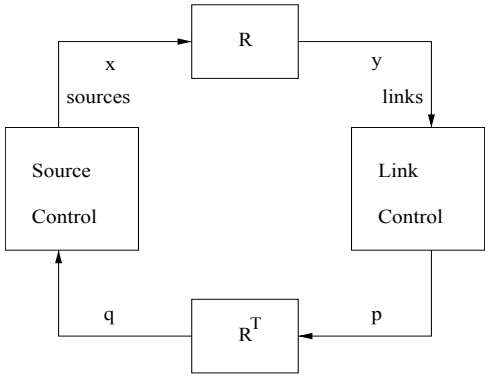

Figure 1.8: A block diagram view of the congestion control algorithm. The controller at the source uses congestion feedback from the link to perform its action.

Proof Differentiating $V(.)$, we get

$$\dot{V} = -\sum_{r \in \mathcal{S}} \frac{\partial V}{\partial x_r} \dot{x}_r = -\sum_{r \in \mathcal{S}} k_r(x_r) \left(U_r'(x_r) - q_r \right)^2 < 0, \ \forall x \neq \hat{x}, \tag{1.29}$$

and $\dot{V} = 0$ if $x = \hat{x}$. Thus, all the conditions of the Lyapunov theorem are satisfied and we have proved that the system state converges to $\hat{x}$, starting from any initial condition.

□

In the proof of the above theorem, we have assumed that the utility, price and scaling functions are such that $W(x)$ has some desired properties. It is very easy to find functions that satisfy these properties. For example, if $U_r(x_r) = w_r \log(x_r)$, and $k_r(x_r) = x_r$, then the primal congestion control algorithm for source r becomes

$$\dot{x}_r = w_r - x_r \sum_{l: l \in r} f_l(y_l),$$

and thus the unique equilibrium point is $w_r/x_r = \sum_{l: l \in r} f_l(y_l)$. If $f_l(.)$ is any polynomial function, then $W(x)$ goes to $-\infty$ as $||x|| \to \infty$ and thus, $V(x) \to \infty$ as $||x|| \to \infty$.

1.4.1 Price Functions and Congestion Feedback

We had earlier argued that collecting the price information from the network is simple. If there is a field in the packet header to store price information, then each link on the route of a packet simply adds its price to this field, which is then echoed back to the source by the receiver in the acknowledgment packet. However, packet headers in the Internet are already crowded with a lot of other information, so Internet practitioners do not like to add many bits in the packet header to collect congestion information. Let us consider the extreme case where there is only one bit available in the packet header to collect congestion information. How could we use this bit to collect the price of route? Suppose that each packet is *marked* with probability $1 - e^{-p_l}$ when the

packet passes through link l. Marking simply means that a bit in the packet header is flipped from a 0 to a 1 to indicate congestion. Then, along a route r, a packet is marked with probability

$$1 - e^{\sum_{l:l\in r} p_l}.$$

If the acknowledgment for each packet contains one bit of information to indicate if a packet is marked or not, then by computing the fraction of marked packets, the source can compute the route price $\sum_{l:l\in r} p_l$. The assumption here is that x_r's change slowly so that each p_l remains roughly constant over many packets. Thus, one can estimate p_l reasonably accurately.

Another price function of interest is found by considering packet dropping instead of packet marking. If packets are dropped due to the fact that a link buffer is full when a packet arrives at the link, then such a dropping mechanism is called a *Droptail* scheme. A crude approximation to the drop probability (also known as packet loss rate) at link l is $\left(\frac{y_l - c_l}{y_l}\right)^+$, which is non-zero only if $y_l = \sum_{r:l\in r} x_r$ is larger than c_l. When packets are dropped at a link for source r, then the arrival rate from source r at the next link on the route would be smaller due to the fact that dropped packets cannot arrive at the next link. Thus, the arrival rate is "thinned" as we traverse the route. However, this is very difficult to model in our optimization framework. Therefore, we assume that the drop probabilities are small so that arrival rate of packets from a given source is approximately the same at all links on its route. Further, the end-to-end drop probability on a route can be approximated by the sum of the drop probabilities on the links along the route if the drop probability at each link is small. Thus, the optimization formulation approximates reality under these assumptions.

1.5 Distributed Algorithms: Dual Solution

In this section we consider another distributed algorithm based on the dual formulation of the utility maximization problem. Consider the resource allocation problem that we would like to solve

$$\max_{x_r} \sum_{r\in\mathcal{S}} U_r(x_r) \tag{1.30}$$

subject to the constraints

$$\sum_{r:l\in r} x_r \le c_l, \ \forall l \in \mathcal{L}, \tag{1.31}$$

$$x_r \ge 0, \ \forall r \in \mathcal{S}. \tag{1.32}$$

The Lagrange dual of the above problem is obtained by incorporating the constraints into the maximization by means of Lagrange multipliers as follows:

$$D(p) = \max_{\{x_r\ge 0\}} \sum_r U_r(x_r) - \sum_l p_l \left(\sum_{s:l\in s} x_s - c_l\right) \tag{1.33}$$

Here the p_ls are the Lagrange multipliers that we saw in Section 1.1. The dual problem may then be stated as

$$\min_{p\ge 0} D(p).$$

As in the case of the primal problem, we would like to design an algorithm which ensures that all the source rates to converge to the optimal solution. Notice that in this case we are looking for a gradient descent (rather than a gradient ascent that we saw in the primal formulation), since we would like to minimize $D(p)$. To find the direction of the gradient, we need to know $\frac{\partial D}{\partial p_l}$.

We first observe that in order to achieve the maximum in (1.33), x_r must satisfy

$$U_r'(x_r) = q_r, \tag{1.34}$$

where, as usual, $q_r = \sum_{l:l\in r} p_l$, is the price of a particular route r. Note that we have assumed that $x_r > 0$ in writing down (1.34). This would be true, for example, if the utility function is an α-utility function with $\alpha > 0$. Now,

$$\begin{aligned}
\frac{\partial D}{\partial p_l} &= \sum_r U_r'(x_r)\frac{\partial x_r}{\partial p_l} - (y_l - c_l) - \sum_k p_k \frac{\partial y_k}{\partial p_l} \\
&= \sum_r U_r'(x_r)\frac{\partial x_r}{\partial p_l} - (y_l - c_l) - \sum_k p_k \sum_{r:k\in r} \frac{\partial x_r}{\partial p_l} \\
&= \sum_r U_r'(x_r)\frac{\partial x_r}{\partial p_l} - (y_l - c_l) - \sum_r \frac{\partial x_r}{\partial p_l} \sum_{k:k\in r} p_k \\
&= \sum_r U_r'(x_r)\frac{\partial x_r}{\partial p_l} - (y_l - c_l) - \sum_r \frac{\partial x_r}{\partial p_l} q_r.
\end{aligned}$$

Thus, using (1.34), we have

$$\frac{\partial D}{\partial p_l} = -(y_l - c_l). \tag{1.35}$$

Recalling that, to minimize $D(p)$, we have to *descend* down the gradient, from (1.34) and (1.35), we have the following dual control algorithm:

$$x_r = U_r'^{-1}(q_r) \quad \text{and} \tag{1.36}$$

$$\dot{p}_l = h_l(y_l - c_l)^+_{p_l}, \tag{1.37}$$

where $h_l > 0$ is a constant and $(g(x))^+_y$ denotes

$$(g(x))^+_y = \begin{cases} g(x), & y > 0, \\ \max(g(x), 0), & y = 0. \end{cases}$$

We use this modification to ensure that p_l never goes negative since we know from the KKT conditions that the optimal price is non-negative. Note that, if $h_l = 1$, the price update above has the same dynamics as the dynamics of the queue at link l. The price increases when the arrival rate is larger than the capacity and decreases when the arrival rate is less than the capacity. Moreover, the price can never become negative. These are exactly the same dynamics that govern the queue size at link l. Thus, one does not even have to explicitly keep track of the price in the dual formulation; the queue length naturally provides this information.

The stability of this algorithm follows in a manner similar to the primal algorithm by considering $D(p)$ as the Lyapunov function since the dual algorithm is simply a gradient algorithm for finding the minimum of $D(p)$.

In the next section, we will discuss practical TCP protocols based on the primal and dual formulations. When we discuss these protocols, we will see that the price functions and congestion control mechanisms obtained from the two formulations have different interpretations.

1.6 Relationship to TCP Protocols

In this section, we explore the relationship between the algorithms discussed in the previous sections and the protocols used in the Internet today. It is important to note that Internet congestion control protocols were *not* designed using the optimization formulation of the resource allocation problem that we have seen in the previous two sections. The predominant concern while designing these protocols was to minimize the risk of congestion collapse, i.e., large-scale buffer overflows, and hence they tended to be rather conservative in their behavior. Even though the current Internet protocols were not designed with clearly-defined fairness and stability ideas in mind, they bear a strong resemblance to the ideas of fair resource allocation that we have discussed so far. In fact, the utility maximization methods presented earlier provide a solid framework for understanding the operation of these congestion control algorithms. Further, going forward, the utility maximization approach seems like a natural candidate framework used to modify existing protocols to adapt to the evolution of the Internet as it continues to grow faster.

As mentioned in the first chapter, the congestion control algorithms used in today's Internet are based on window flow control. The idea is that each user maintains a number called a *window size*, which is the number of unacknowledged packets that it is allowed to send into the network. Any new packet can be sent only when an acknowledgment for one of the previous sent packets is received by the sender as shown in Figure 1.9.

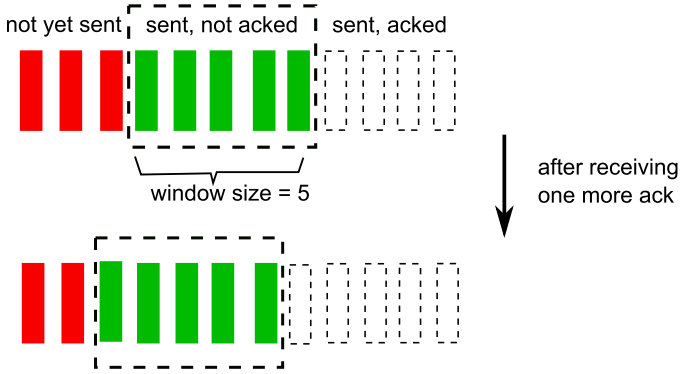

Figure 1.9: Window Flow Control. The window size is set to be 5, so at most five unacknowledged packets are allowed. After an additional acknowledgement is received, one more packet can be sent out.

The window size W is closely related to the data rate x of the flow as explained below. The amount of time that elapses between the sending of a packet and the reception of feedback from the destination is called the Round-Trip Time (RTT). We denote the RTT by T. Assume the link speeds are very fast so that the time that it takes to process a packet at a link is negligible compared to the RTT. Suppose that the window size is W and as a result, the source send W packets into the network. Then, since the processing time is negligible, the acks for these packets will arrive roughly at the same time at the source (after one RTT time). Thus, the average rate of transmission x is just the window size divided by T, i.e., $x = W/T$. Clearly this model is very crude, but it works surprisingly well in practice.

Because of this relation between window size and data rate, the data rate of a flow can be

controlled by adapting the window size. *Transmission Control Protocol* (TCP) is the protocol that determines the increase/decrease of the window size in today's Internet. There are several different flavors of TCP congestion control, each of which operates somewhat differently. But all versions of TCP are window-based protocols. TCP adapts the window size in response to congestion information. The window size is increased if the sender determines that there is excess capacity present in the route, and decreased if the sender determines that the current number of in-flight packets exceeds the capacity of the route. A decision on whether to send a new packet, and whether the window is to be increased or decreased, is taken upon reception of the acknowledgement packet. This means that the decision-making process has no periodicity that is decided by a clock of fixed frequency. TCP is therefore called *self-clocking*. Different versions of TCP use different algorithms to determine when and how to increase/decrease the window sizes. We discuss these versions of TCP next.

1.6.1 TCP-Reno

The most commonly used TCP versions used for congestion control in the Internet today are Reno and NewReno. Both of them are updates of TCP-Tahoe, which was introduced in 1988. Although they vary significantly in many regards, the basic approach to congestion control is similar. The idea is to use successful reception packets as an indication of available capacity and dropped packets as an indication of congestion. We consider a simplified model for the purpose of exposition. Each time the destination receives a packet, it sends an acknowledgement (also called *ack*) asking for the next packet in sequence. For example, when packet 1 is received, the acknowledgement takes the form of a request for packet 2. If, instead of the expected packet 2, the destination receives packet 3, the acknowledgement still requests packet 2. Reception of three duplicate acknowledgments or *dupacks* (i.e., four successive identical acks) is taken as an indication that packet 2 has been lost due to congestion. The source then proceeds to cut down the window size and also to re-transmit lost packets. In case the source does not receive any acknowledgements for a pre-determined time, it assumes that all its packets in flight have been lost and times out.

When a non-duplicate acknowledgment is received, the protocol increases its window size. The amount by which the window size is increased depends upon the TCP transmission *phase*. TCP operates in two distinct phases. When file transfer begins, the window size is 1, but the source rapidly increases its transmission window size so as to reach the available capacity quickly. Let us denote the window size by W. The algorithm increases the window size by 1 each time an acknowledgement is received, i.e., $W \leftarrow W + 1$. This is called the *slow-start* phase. Since we have assumed that packet processing time at a link is small compared to the RTT, the number of acknowledgements received by a source in one RTT would be approximately equal to the window size. If we increase the window size by one for each successful packet transmission, this also means that (if all transmissions are successful) the window would roughly double in each RTT, so we have an exponential increase in rate as time proceeds. Slow-start refers to the fact that the window size is still small in this phase, but the rate at which the window increases is quite rapid. When the window size either hits a threshold, called the *slow-start threshold or ssthresh* or if a packet loss is detected (immediately leading to a halving of window size), the algorithm shifts to a more conservative window-increase algorithm called the *congestion avoidance* phase. When in the congestion-avoidance phase, the algorithm increases the window size by $1/W$ every time feedback of a successful packet transmission is received, so we now have $W \leftarrow W + 1/W$. Thus, in

each RTT, the window increases by one packet, i.e., a linear increase in rate as a function of time. When a packet loss is detected by the receipt of three dupacks, the slow-start threshold (ssthresh) is set to $W/2$ and TCP Reno cuts its window size by half, i.e., $W \leftarrow W/2$. Protocols of this sort where the increment is by a constant amount, but the decrement is by a multiplicative factor are called *additive-increase multiplicative-decrease* (AIMD) protocols. When packet loss is detected by a time-out, the window size is reset to 1 and TCP enters the slow-start phase. We illustrate the operation of TCP-Reno in Figure 1.10.

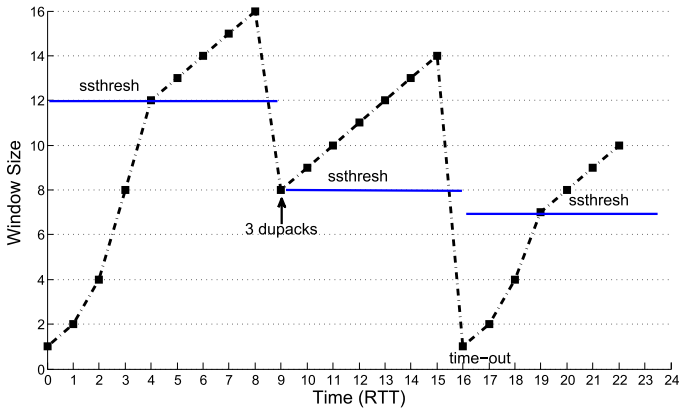

Figure 1.10: Operation of TCP-Reno. The window size first exponentially increases during the slow start phase until reaching ssthresh (=12). Then TCP-Reno enters congestion avoidance phase and the window size linearly increases. When a loss is detected by receiving three dupacks at the 9^{th} RTT, the ssthresh is set to be 8, and the window size is set to 8 and increases linearly. When a time-out occurs at the 16^{th} RTT, the ssthresh is set to be 7. The window size is set to be 1, and TCP-Reno enters the slow-start phase.

Now, the slow-start phase of a flow is relatively insignificant if the flow consists of a large number of packets. So we will consider only the congestion-avoidance phase. Let us call the congestion window at time t as $W(t)$. This means that the number of packets in-flight is $W(t)$. The time taken by each of these packets to reach the destination, and for the corresponding acknowledgement to be received is T. The RTT is a combination of propagation delay and queueing delay, but we ignore the fluctuations in queueing delay and assume that the RTT is a constant.

Let us now write down TCP Reno's behavior in terms of the differential equation models. Consider a flow r. As defined above, let $W_r(t)$ denote the window size and T_r its RTT. Earlier we used the notation $q_r(t)$ to denote the price of a route r. TCP uses packet loss probability as the price of a route. So we use the same notation $q_r(t)$ to denote the packet loss probability under TCP. We can model the congestion avoidance phase of TCP-Reno as

$$\dot{W}_r(t) = \frac{x_r(t-T_r)(1-q_r(t))}{W_r(t)} - \beta x_r(t-T_r)q_r(t)W_r(t). \tag{1.38}$$

The above equation can be derived as follows:

- The rate at which the source obtains acknowledgements is $x_r(t-T_r)(1-q_r(t))$. Since each acknowledgement leads to an increase by $1/W_r(t)$, the rate at which the window size increases is given by the first term on the right side.

- The rate at which packets are lost is $x_r(t - T_r)q_r(t)$. Such events would cause the window size to be decreased by a factor that we call β. This is the second term on the right side. Considering the fact that there is a halving of window size due to loss of packets, β would be naturally taken to be 1/2. However, studies show that a more precise value of β when making a continuous-time approximation of TCP's behavior is close to 2/3.

To compare the TCP formulation above to the resource allocation framework, we write $W_r(t)$ in terms of $x_r(t)$ ($W_r(t) = x_r(t)T$) which yields

$$\dot{x}_r = \frac{x_r(t - T_r)(1 - q_r(t))}{T_r^2 x_r(t)} - \beta x_r(t - T_r)q_r(t)x_r(t). \tag{1.39}$$

The equilibrium value of x_r is found by setting $\dot{x}_r = 0$, and is seen to be

$$\hat{x}_r = \sqrt{\frac{1 - \hat{q}}{\beta\hat{q}}}\frac{1}{T_r},$$

where $\hat{q}_r$ is the equilibrium loss probability. For small values of $\hat{q}_r$ (which is what one desires in the Internet),

$$\hat{x}_r \propto 1/(T_r\sqrt{\hat{q}_r}),$$

i.e., the equilibrium rate of TCP-rate is inversely proportional to the RTT and the square-root of the loss probability. This result is well-known and widely used in the performance analysis of TCP-Reno.

Relationship with Primal Algorithm

Now, consider the controller (1.39) again. Suppose that there were no feedback delay, but the equation is otherwise unchanged. So T_r^2 that appears in (1.39) is just some constant now. Also, let $q_r(t)$ be small, i.e., the probability of losing a packet is not too large. Then the controller reduces to

$$\begin{aligned} \dot{x}_r &= \frac{1}{T_r^2} - \beta x_r^2 q_r \\ &= \beta x_r^2 \left(\frac{1}{\beta T_r^2 x_r^2} - q_r \right). \end{aligned}$$

Comparing the above equation with the primal congestion controller (1.23), we find that the utility function of source r satisfies

$$U_r'(x_r) = \frac{1}{\beta T_r^2 x_r^2}.$$

We can find the source utility (up to an additive constant) by integrating the above, which yields

$$U_r(x_r) = -\frac{1}{\beta T_r^2 x_r}.$$

Thus, TCP-Reno can be approximately viewed as a control algorithm that attempts to achieve weighted minimum potential delay fairness.

If we do not assume that q_r is small, the delay-free differential equation is given by

$$\begin{aligned}\dot{x}_r &= \frac{1-q_r}{T_r^2} - \beta x_r^2 q_r \\ &= (\beta x_r^2 + 1/T_r^2)\left(\frac{1}{\beta x_r^2 + \frac{1}{T_r^2}}\frac{1}{T_r^2} - q_r\right),\end{aligned}$$

Thus,

$$U_r'(x_r) = \frac{1}{T_r^2}\frac{1}{\beta x_r^2 + \frac{1}{T_r^2}} \qquad \Rightarrow \qquad U_r(x_r) = \frac{1}{T_r\sqrt{\beta}}\tan^{-1}\left(\sqrt{\beta}\,T_r x_r\right),$$

where the utility function is determined up to an additive constant.

A Generalization of TCP-Reno

Instead of increasing the window size by $1/W$ for each ack and halving upon detecting a loss, one could consider other increase-decrease choices as well. Consider a protocol where $W \leftarrow W + a\ W^n$ when an acknowledgement is received, while a loss triggers a window decrease given by $W \leftarrow W - b\ W^m$. Setting $a = 1,\ n = -1,\ b = 0.5$, and $m = 1$ would yield TCP-Reno type behavior. The equivalent rate-based equation describing the dynamics of such a protocol would be

$$\dot{x}_r = \frac{x_r(t-T_r)}{T_r}\left(a(x_r(t)\ T_r)^n(1-q_r(t)) \ - \ b(x_r(t)\ T_r)^m q_r(t)\right). \tag{1.40}$$

Ignoring the feedback delay in obtaining the congestion information, the above differential equation becomes

$$\dot{x}_r = \frac{x_r}{T_r}\left(a(x_r\ T_r)^n(1-q_r(t)) - \ b(x_r\ T_r)^m q_r\right),$$

which can be rewritten as

$$\dot{x}_r = (a x_r^{n+1} T_r^{n-1})\left(1 + \frac{b}{a}(x_r T_r)^{m-n}\right)\left(\frac{1}{1+\frac{b}{a}(x_r T_r)^{m-n}} - q_r\right).$$

Note that

$$\frac{1}{1+\frac{b}{a}(x_r T_r)^{m-n}}$$

is a decreasing function (thus, its derivative will be negative) if $m > n$ and hence one can view the above differential equation as the congestion controller for source r with a concave utility function

$$\int_0^{x_r} \frac{1}{1+\frac{b}{a}(xT_r)^{m-n}}dx.$$

Different choices of a, b, m, and n have been studied for various applications, but so far TCP-Reno continues to be the most dominant form of TCP and therefore, we do not discuss these variants here.

1.6.2 TCP-Vegas: A Delay Based Algorithm

We now consider another variation of TCP called TCP-Vegas. TCP-Vegas uses queueing delay, instead of packet loss as TCP-Reno, to infer congestion in the network. The idea is to first identify the propagation delay of the route by assuming that it is equal to the smallest RTT seen by the source. This is a reasonable assumption if we assume that queues empty occasionally in which case the only delay is the propagation delay. Let us denote the estimated propagation delay by T_p. Any excess delay above this amount would be queueing delay and we denote it by T_q. The objective of the algorithm is to calculate the value of window size such that the number of packets in the network is small. When this occurs, the rate of generation of packets is equal to the available capacity on the route. We now study the details of this algorithm.

If there is no queueing delay, the throughput would be approximately given by

$$e = \frac{W}{T_p},$$

where W is the window size. However, the actual throughput is the number of packets that make it successfully to the destination in a fixed amount of time. To calculate this quantity, the source sends a marked packet and waits for it to be acknowledged. The duration of time that it takes for this event to occur is the round-trip time $T_p + T_q$. Suppose during this time, the source receives S acknowledgments, then the actual throughput is estimated as

$$a = \frac{S}{T_p + T_q}.$$

Whenever we receive the acknowledgment for a marked packet at some time t, we have two values—the expected throughput $e(t)$ and the actual throughput $a(t)$. If $a(t)$ is less than $e(t)$, it means that our transmission rate is too high, so we should cut down the window size. On the other hand, if $a(t)$ is greater than $e(t)$, it means that the estimate of the available rate is too low and we should increase the window size. Formally, we define constants α and β, with $\alpha \leq \beta$ and proceed as follows:

- if $\alpha \leq (e(t) - a(t)) \leq \beta$, then do nothing. This means that our estimate of the throughput is fairly accurate, and everything is as it should be.
- if $(e(t) - a(t)) > \beta$, decrease the window size by 1 for the next RTT. This means that our estimate is too high and the window size must be reduced.
- if $(e(t) - a(t)) < \alpha$, increase the window size by 1 for the next RTT. This means that our estimate is too low and the network can support more than we think.

Note that both the increase and decrease of window size are linear in nature. Also, the algorithm uses the usual slow start mechanism, with exponential growth initially. The behavior of TCP-Vegas under ideal conditions would look something like Figure 1.11.

Next, we interpret TCP-Vegas as a resource allocation algorithm in the utility maximization framework. We assume $\alpha = \beta$ and the propagation delay is estimated accurately, i.e., for source r, $T_{pr}(t) \equiv T_{pr}$ for all t.

At equilibrium, the estimated throughput is the same as the actual throughput, with the window size and the number of acknowledgements received in an RTT being the same. If we denote

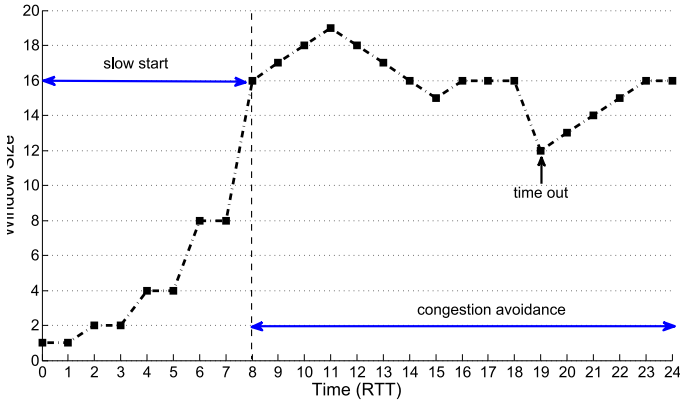

Figure 1.11: Operation of TCP-Vegas. It preserves the slow-start phase, but doubles the window size every other RTT to have a accurate comparison between the actual throughput and expected throughput. After switching to congestion avoidance mode, the window increases and decreases linearly. Ideally, the algorithm should converge to a stable window size (e.g., during the 17^{th} and 18^{th} RTTs). When a time-out occurs (during the 19^{th} RTT), the window size is reduced by 1/4.

the equilibrium window size and queueing delay of source r by $\hat{W}_r$ and $\hat{T}_{qr}$ respectively (and by assumption, the propagation delay T_{pr} is correctly estimated), then

$$\frac{\hat{W}_r}{T_{pr}} - \frac{\hat{W}_r}{T_{pr} + \hat{T}_{qr}} = \alpha. \tag{1.41}$$

At equilibrium, the transmission rate $\hat{x}$ is approximately

$$\hat{x}_r = \hat{W}_r/(T_{pr} + \hat{T}_{qr}),$$

which means that (1.41) can be simplified to

$$\frac{\alpha T_{pr}}{\hat{x}_r} = \hat{T}_{qr}. \tag{1.42}$$

Now that we know what the equilibrium transmission rate looks like, let us study what the equilibrium queueing delay $\hat{T}_{qr}$ would look like. If the queue length at equilibrium is denoted by $\hat{b}_l$, then the equilibrium queueing delay at link l is $\hat{b}_l/c_l$ (where c_l is the capacity of link l). So we have

$$\hat{T}_{qr} = \sum_{l:l\in r} \frac{\hat{b}_l}{c_l}. \tag{1.43}$$

Also, if

$$\sum_{k:l\in k} \hat{x}_k < c_l,$$

then there is no queueing delay, i.e., $\hat{b}_l = 0$ in this case. Note that since the aggregate equilibrium transmission rate of all flows using link l cannot exceed the link capacity, we cannot possibly have

that

$$\sum_{k:l\in k} \hat{x}_k > c_l.$$

Thus, if $\hat{b}_l \neq 0$, it means that

$$\sum_{k:l\in k} \hat{x}_k = c_l.$$

Thus, we have from the above and (1.42), that the equilibrium conditions are

$$\frac{\alpha T_{pr}}{\hat{x}_r} = \sum_{l:l\in r} \frac{\hat{b}_l}{c_l},$$

with

$$\frac{\hat{b}_l}{c_l}\left(\sum_{k:l\in k} \hat{x}_k - c_l\right) = 0 \quad \forall\ l.$$

These are the KKT conditions for the utility maximization problem

$$\max_{\{x_r\}} \alpha T_{pr} \log x_r, \tag{1.44}$$

subject to

$$\sum_{r:l\in r} x_r \leq c_l, \quad \forall\ l$$
$$x_r \geq 0, \forall r$$

with $\frac{\hat{b}_l}{c_l}$ as the Lagrange multipliers. Thus, assuming it converges to some equilibrium rates, TCP-Vegas is weighted-proportionally fair. If we let each flow have a different value of α, i.e., we associate α_r with route r, then the equilibrium rates will maximize $\sum_r \alpha_r T_{pr} \log x_r$. Recall that we have assumed that $\alpha_r = \beta_r$ in this analysis.

Relation to Dual Algorithms and Extensions

We now consider the relationship between TCP-Vegas and dual algorithms. A weighted proportionally fair dual algorithm would use the controller obtained by substituting $w_r \log(x_r)$ as the utility function in (1.36) and (1.37), which yields

$$x_r = \frac{w_r}{q_r} \quad \text{and} \tag{1.45}$$
$$\dot{p}_l = h_l(y_l - c_l)^+_{p_l}. \tag{1.46}$$

To avoid confusion, we note that w_r is the weight assigned to source r and is unrelated to the window size W_r. If we choose $h_l = \frac{1}{c_l}$, then the price function of a link becomes the queueing delay experienced by packets using that link, which, when added to a constant propagation delay, is the feedback that is used in the TCP-Vegas algorithm.

Let us study the source rates achieved in TCP-Vegas more closely to create a fluid model equivalent. From the algorithm description (with $\alpha = \beta$), TCP-Vegas updates its window size W_r based on whether

$$\frac{W_r}{T_{pr}} - \frac{W_r}{T_{pr} + T_{qr}} < \alpha \quad \text{or} \quad \frac{W_r}{T_{pr}} - \frac{W_r}{T_{pr} + T_{qr}} > \alpha. \tag{1.47}$$

Using the approximation $x_r = \frac{W_r}{T_{pr}+T_{qr}}$, we can rewrite the conditions as

$$x_r \; T_{qr} < \alpha T_{pr} \quad \text{or} \quad x_r \; T_{qr} > \alpha T_{pr}. \tag{1.48}$$

As in (1.43), we also have

$$T_{qr} = \sum_{l:l\in r} \frac{b_l}{c_l} = \sum_{l:l\in r} p_l, \tag{1.49}$$

where b_l is the queue length at link l, and we have used p_l to denote the queueing delay at link l, which acts as the price function for TCP-Vegas. Combining the above expressions, the condition for increase/decrease becomes

$$x_r \sum_{l:l\in r} p_l < \alpha T_{pr} \quad \text{or} \quad x_r \sum_{l:l\in r} p_l > \alpha T_{pr}, \tag{1.50}$$

So the window control algorithm can be written as

$$W_r \leftarrow \left[W_r + \frac{1}{T_{pr} + T_{qr}} \; \text{sgn}\left(\alpha T_{pr} - x_r \; T_{qr}\right) \right]^+_{W_r}, \tag{1.51}$$

where $\text{sgn}(z) = -1$ if $z < 0$, $\text{sgn}(z) = 1$ if $z > 0$, and $\text{sgn}(z) = 0$ if $z = 0$. Thus, we can now write down the differential equations describing TCP-Vegas as

$$\dot{p}_l = \frac{1}{c_l}(y_l - c_l)^+_{p_l} \tag{1.52}$$

$$\dot{W}_r = \left[\frac{1}{T_{pr} + T_{qr}} \; \text{sgn}\left(\alpha T_{pr} - x_r \; T_{qr}\right) \right]^+_{W_r} \tag{1.53}$$

$$x_r = \frac{W_r}{T_{pr} + T_{qr}}, \tag{1.54}$$

with $T_{qr} = \sum_{l:l\in r} p_l$. The above is not the same as the dual algorithm that we derived in the previous section. However, the price update dynamics are the same as the price update for the dual algorithm. Further, at the source, by attempting to increase or decrease the rate based on whether x_r is less than or greater than $\alpha T_{pr}/T_{qr}$, it is clear the source attempts to drive the system towards

$$x_r = \frac{\alpha T_{pr}}{T_{qr}} = \frac{\alpha T_{pr}}{\sum_{l\in r} p_l},$$

which is the desired source behavior for a dual congestion controller. Thus, one can interpret TCP-Vegas as an algorithm that approximates the dual congestion control algorithm.

A modification of TCP-Vegas called FAST-TCP has been suggested for very high-speed networks. In FAST-TCP, the window size is increased or decreased depending upon how far the window size is from a desired equilibrium point. The fluid model describing the protocol is

$$\dot{p}_l = \frac{1}{c_l}(y_l - c_l)^+_{p_l} \tag{1.55}$$

$$\dot{W}_r = \gamma_r \left(\alpha_r - x_r \; T_{qr}\right)^+_{W_r} \tag{1.56}$$

$$x_r = \frac{W_r}{T_{pr} + T_{qr}}, \tag{1.57}$$

where α_r determines the desired equilibrium point and γ_r is a scaling constant. Replacing the sgn function in TCP-Vegas with the difference between the current operating point and the desired equilibrium allows FAST-TCP to rapidly approach the desired equilibrium point.

Chapter 2

Links: Statistical Multiplexing and Queues

In Chapter 1, we assumed that the transmission rates x_r are positive, and derived fair and stable resource allocation algorithms. In reality, since data is transmitted in the form of packets, the rates x_r are converted to discrete window sizes, which results in bursty (non-smooth) arrival rates at the links in the network. In addition, many flows in the Internet are very short (consisting of only a few packets) for whom the convergence analysis in the previous chapter does not apply. Further, there may also be flows which are not congestion controlled. We address these deviations from our basic model by considering packet arrivals at links to be random processes. In this chapter, for such random arrival processes, we are interested in answering the following questions: *what are the buffer sizes required to temporarily store bursty packet arrivals before transmission over a link; what is the relationship between buffer overflow probabilities, delays and the burstiness of the arrival processes; how do we provide isolation among flows so that each flow is guaranteed a minimum rate at a link, independent of the burstiness of the other flows sharing the link?*

2.1 Mathematical Background: The Chernoff Bound

In this section, we present the Chernoff bound, which provides a bound on the tail distribution of the sum of independent random variables.

Lemma 2.1.1 (Markov's Inequality) *For a positive random variable X, the following inequality holds for any $\epsilon > 0$:*

$$\Pr(X \geq \epsilon) \leq \frac{E(X)}{\epsilon}.$$

Proof Define a random variable Y such that $Y = \epsilon$ if $X \geq \epsilon$ and $Y = 0$ otherwise. So

$$E[X] \geq E[Y] = \epsilon \Pr(X \geq \epsilon).$$

□

Theorem 2.1.2 (The Chernoff Bound) *Consider a sequence of independently and identically distributed (i.i.d.) random variables $\{X_i\}$ with mean $\mu = E[X_i]$. For any constant x, the following inequality holds:*

$$\Pr\left(\sum_{i=1}^{n} X_i \geq nx\right) \leq \exp\left(-n \sup_{\theta \geq 0}\{\theta x - \log M(\theta)\}\right), \tag{2.1}$$

where $M(\theta) \triangleq E\left[e^{\theta X_1}\right]$ is the moment generating function of X_1.

If $\{X_i\}$ are Bernoulli random variables with parameter μ, and $x > 0$, then

$$\Pr\left(\sum_{i=1}^{n} X_i \geq n(\mu + x)\right) \leq \exp\left(-nD((\mu + x)\|\mu)\right), \tag{2.2}$$

where

$$D((\mu + x)\|\mu) = (\mu + x)\log\frac{\mu + x}{\mu} + (1 - \mu - x)\log\frac{1 - \mu - x}{1 - \mu}$$

is the Kullback-Leibler distance between Bernoulli random variables with parameter $\mu + x$ and parameter μ.

Proof For any $\theta \geq 0$, we have

$$\begin{aligned}\Pr\left(\sum_{i=1}^{n} X_i \geq nx\right) &\leq \Pr(e^{\theta\sum_{i=1}^{n} X_i} \geq e^{\theta nx}) \\ &\leq \frac{E\left[e^{\theta\sum_{i=1}^{n} X_i}\right]}{e^{\theta nx}},\end{aligned} \tag{2.3}$$

where the first inequality becomes an equality if $\theta > 0$ and the second inequality follows from the Markov inequality. Since inequality (2.3) holds for all $\theta \geq 0$, we further obtain

$$\begin{aligned}\Pr(\sum_{i=1}^{n} X_i \geq nx) &\leq \inf_{\theta \geq 0} \frac{E\left[e^{\theta\sum_{i=1}^{n} X_i}\right]}{e^{\theta nx}} \\ &= e^{-n\sup_{\theta\geq 0}\{\theta x - \log M(\theta)\}}.\end{aligned} \tag{2.4}$$

Recall that $M(\theta) \triangleq E\left[e^{\theta X_i}\right]$ is the moment generating function of X_i. Inequality (2.4) is called the Chernoff bound.

Inequality (2.2) holds because

$$\sup_{\theta \geq 0}\{\theta x - \log M(\theta)\} = D((\mu + x)\|\mu)$$

when $\{X_i\}$ are Bernoulli random variables and $x > 0$.

□

2.2 Statistical Multiplexing and Packet Buffering

When multiple data sources share the same link, the bandwidth of the link needs to be allocated properly. To guarantee no packet loss and small transmission latencies, one may allocate the bandwidth according to data sources' peak transmission rates. For example, if a data source has a peak rate of R bits/second, then a bandwidth of R bits/second of the link is reserved for that source. This approach will provide very good Quality of Services (QoSs) to data sources in terms of bandwidth, delay and jitter, but could cause the link to be under-utilized since the typical total data rate of the sources may be much smaller than the sum of their peak rates.

The bandwidth allocated to sources is often much smaller than the sum of their peak rates, and is slightly larger than the sum of the average rates of the sources. The link then relies on the fact that the probability that the sum of rates exceeds the sum of the average rates is small. This type of resource allocation is called *statistical multiplexing.* Compared to bandwidth allocation based on peak rates, statistical multiplexing allows a link to support a larger number of data sources, as shown in a simple example below.

Example 2 *Consider a link with bandwidth 10 Mbps, which is shared by multiple data sources. At any given time, a source is active with a probability of* 0.1, *and transmits at a rate of* 100 *Kbps when active.*

If the link bandwidth is allocated according to the peak rate, then the link needs to reserve 100 *Kbps for each source. In this case, the maximum number of sources that can be allowed is given by*

$$\frac{10\ Mbps}{100\ Kbps} = 100.$$

Now consider statistical multiplexing and assume there are n sources using the link. Define X_i *to be a random variable such that* $X_i = 1$ *if source i is active and* $X_i = 0$ *otherwise. We apply the Chernoff bound for Bernoulli random variables to bound the probability that the aggregated rate of active sources exceeds the link capacity, i.e., the following overflow probability*

$$\Pr\left(\sum_{i=1}^{n} X_i \geq 100\right). \tag{2.5}$$

The result is illustrated in Figure 2.1.

We observe that the link can accommodate up to 750 *sources if the overflow probability is allowed to be* 0.01%. *We can increase the number of sources to* 800 *if the overflow probability is allowed to be* 1%. *Thus, statistical multiplexing can dramatically increase network capacity at the cost of very small loss probabilities.* □

2.2.1 Queue Overflow

In Example 2, we have seen that statistical multiplexing performs well even without buffer. In practice, when the number of arrivals exceeds the link capacity, the packets will first be stored in a buffer, instead of being dropped immediately. The focus of the rest of this chapter is to understand the behavior of the buffer under various arrival processes and buffer models. As a starting point, we assume the buffer is of infinite size and calculate the probability that the amount of buffered packets exceeds a certain threshold B.

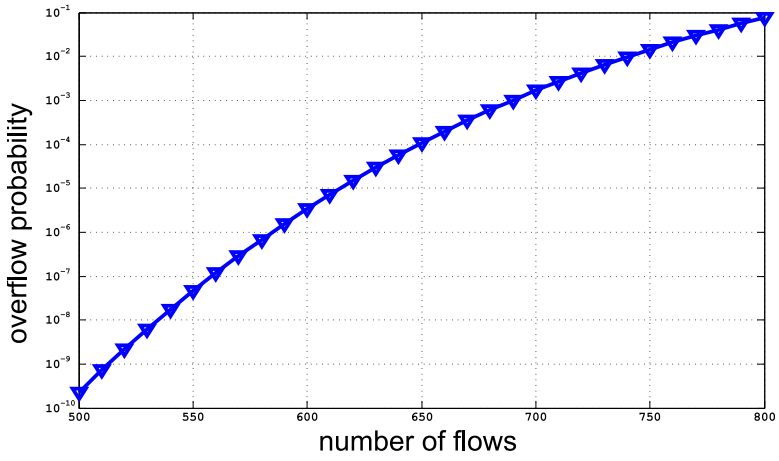

Figure 2.1: The overflow probabilities versus the number of sources in the network

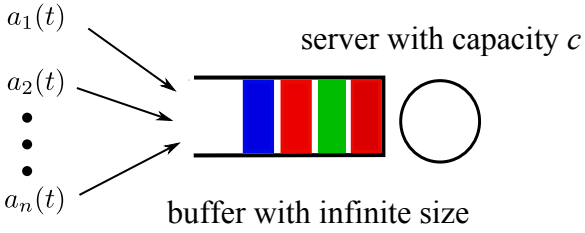

Figure 2.2: A queue shared by n sources

We model a single link shared by n sources as a discrete time queueing system with a single server and infinite buffer space, as shown in Figure 2.2. The server represents the link and can serve c packets per time slot. We assume packets are of the same size here, and will study varying packet sizes later. We define $a_i(t)$ to be the number of packets injected by source i in time slot t, and assume $a_i(t)$ are i.i.d. across time and sources. Further, assume $\lambda \triangleq E[a_i(t)] < c/n$, so the overall arrival rate is less than the link capacity.

Let $q(t)$ denote the number of buffered packets (queue length) time t. We assume packets arrive at the beginning of each time slot and depart at the end of each time slot, and $q(t)$ is measured before packet arrivals. The queue evolution can be described as follows:

$$q(t+1) = (q(t) + a(t) - c)^+,$$

where $a(t) = \sum_{i=1}^{n} a_i(t)$ and $x^+ = \max\{0, x\}$. Assuming the system starts from $t = 0$ with empty queue, i.e., $q(0) = 0$, we next consider the queue length at time k. According to the definition of x^+, we first have

$$q(k) = (q(k-1) + a(k-1) - c)^+ = \max\{q(k-1) + a(k-1) - c, 0\},$$

and

$$q(k-1) = \max\{q(k-2) + a(k-2) - c, 0\}.$$

Substituting the second equation into the first one, we obtain

$$\begin{aligned} q(k) &= \max\{\max\{q(k-2)+a(k-2)-c,0\}+a(k-1)-c,0\} \\ &= \max\{q(k-2)+a(k-1)+a(k-2)-2c, a(k-1)-c, 0\}. \end{aligned}$$

Recursively applying the procedure above, we get

$$q(t) = \max_{t \geq k \geq 1} \left(\sum_{s=1}^{k} a(t-s) - kc \right). \tag{2.6}$$

Next we compute the probability that the queue length at time t exceeds B for some constant B. Based on equation (2.6), we have

$$\Pr(q(t) \geq B) = \Pr\left(\max_{t \geq k \geq 1} \left(\sum_{s=1}^{k} a(t-s) - kc \right) \geq B \right).$$

Note that

$$\Pr\left(\max_{t \geq k \geq 1} \left(\sum_{s=1}^{k} a(t-s) - kc \right) \geq B \right) = \Pr\left(\bigcup_{k=1}^{t} \left(\sum_{s=1}^{k} a(t-s) - kc \geq B \right) \right).$$

Following the union bound,

$$\begin{aligned} \Pr\left(\max_{t \geq k \geq 1} \left(\sum_{s=1}^{k} a(t-s) - kc \right) \geq B \right) &\leq \sum_{k=1}^{t} \Pr\left(\sum_{s=1}^{k} a(t-s) - kc \geq B \right) \\ &= \sum_{k=1}^{t} \Pr\left(e^{\theta \sum_{s=1}^{k} a(t-s)} \geq e^{\theta(kc+B)} \right) \\ &\leq \sum_{k=1}^{t} E\left[e^{\theta \sum_{s=1}^{k} a(t-s)} \right] e^{-\theta(kc+B)}, \end{aligned}$$

where the last inequality follows the Markov inequality.

Since $a_i(t)$ are i.i.d. across time slots and sources,

$$E\left[e^{\theta \sum_{s=1}^{k} a(t-s)} \right] = \prod_{s=1}^{k} \prod_{i=1}^{n} E\left[e^{\theta a_i(t-s)} \right] = M(\theta)^{nk},$$

where $M(\theta) = E[e^{\theta a_i(t)}]$ is the moment generating function. Defining $\Lambda(\theta) = \log M(\theta)$, we obtain the following upper bound:

$$\Pr(q(t) \geq B) \leq \sum_{k=1}^{t} e^{nk\Lambda(\theta)} e^{-\theta(kc+B)}, \tag{2.7}$$

which leads to the following theorem that shows that the probability of queue overflow decreases exponentially as B increases.

Theorem 2.2.1 *For all $\theta > 0$ such that $\frac{\Lambda(\theta)}{\theta} < \frac{c}{n}$, the queue overflow probability satisfies*

$$\Pr(q(t) \geq B) \leq \frac{e^{n\Lambda(\theta)-\theta c}}{1 - e^{n\Lambda(\theta)-\theta c}} e^{-\theta B}. \tag{2.8}$$

Proof Note that upper bound (2.7) can be rewritten as

$$\Pr(q(t) \geq B) \leq \sum_{k=1}^{t} e^{-k(\theta c - n\Lambda(\theta))} e^{-\theta B} \leq e^{-\theta B} \sum_{k=1}^{\infty} e^{k(n\Lambda(\theta)-\theta c)}. \tag{2.9}$$

When $\frac{\Lambda(\theta)}{\theta} < \frac{c}{n}$ holds, we have $e^{(n\Lambda(\theta)-\theta c)} < 1$ which implies that

$$\sum_{k=1}^{\infty} e^{k(n\Lambda(\theta)-\theta c)} = \frac{e^{n\Lambda(\theta)-\theta c}}{1 - e^{n\Lambda(\theta)-\theta c}}.$$

So the theorem holds.

□

From the above result, we see that the overflow probability decreases exponentially with B with decay rate θ provided that

$$\frac{c}{n} > \frac{\Lambda(\theta)}{\theta}.$$

Theorem 2.2.1 shows that when the link capacity per source is greater than $\Lambda(\theta)/\theta$, the overflow probability deceases exponentially at rate θ when the buffer size B increases. The quantity $\frac{\Lambda(\theta)}{\theta}$ is called the *effective bandwidth of a source.*

We now look at the range of this effective bandwidth. Suppose that $a_i(t)$ take values in a finite set such that

$$a_i(t) \in \{a_1, a_2, ..., a_m\} \quad \text{and} \quad a_j < a_{j+1}.$$

Define

$$p_j \triangleq \Pr(a_i(t) = a_j) > 0 \quad \forall\ j \in \{1, 2, ..., m\}.$$

Note that the larger the θ, more stringent is the QoS requirement. Consider two extreme cases:

- If θ is close to 0, then

$$\begin{aligned} \frac{\Lambda(\theta)}{\theta} &= \frac{\log E\left[e^{\theta a_i(t)}\right]}{\theta} \\ &\approx_{(a)} \frac{\log E[1 + \theta a_i(t)]}{\theta} \\ &= \frac{\log(1 + \theta\lambda)}{\theta} \\ &\to_{(b)} \lambda \qquad \text{as } \theta \to 0, \end{aligned}$$

  where both (a) and (b) can be precisely justified using Taylor's theorem.

- If θ is very large, then

$$\begin{aligned} \frac{\Lambda(\theta)}{\theta} &= \frac{\log\left(\sum_{j=1}^{m} e^{\theta a_j} p_j\right)}{\theta} \\ &\approx_{(c)} \frac{\log\left(e^{\theta a_m} p_m\right)}{\theta} \\ &\to a_m \qquad \text{as } \theta \to \infty, \end{aligned}$$

where approximation (c) holds because $e^{\theta a_j}/e^{\theta a_m} \to 0$ as $\theta \to \infty$ for any $a_j < a_m$.

Thus, the effective bandwidth of the source increases from the mean arrival rate λ to the maximum arrival rate a_m, as the QoS parameter θ becomes more and more stringent.

Besides the overflow probability in Theorem 2.2.1, other performance metrics, such as expected queue length and queueing delay, and the probability of packet loss when the buffer size is finite, are also important in practice. Markov chains and queueing theory will be introduced next for the purpose of quantitatively understanding these performance metrics, at least for simple arrival processes.

2.3 Mathematical Background: Discrete-time Markov Chains

Let $\{X_k\}$ be a discrete-time stochastic process which takes on values in a countable set $\mathcal{S}$, where k is time index, called the state space. $\{X_k\}$ is called a discrete-time Markov chain (or simply a Markov chain, when the discrete nature of the time index is clear) if

$$\Pr(X_k = i_k \mid X_{k-1} = i_{k-1}, X_{k-2} = i_{k-2}, \dots) = \Pr(X_k = i_k \mid X_{k-1} = i_{k-1}),$$

where $i_j \in \mathcal{S}$.

A Markov chain is said to be *time homogeneous* if $\Pr(X_k = j \mid X_{k-1} = i)$ is independent of k. We will only consider time-homogeneous Markov chains here. Associated with each Markov chain is a matrix called the probability transition matrix, denoted by $\mathbf{P}$, whose $(i,j)^{\text{th}}$ element is given by $P_{ij} = \Pr(X_k = j \mid X_{k-1} = i)$. Let $p[k]$ denote a row vector of probabilities with $p_j[k] = \Pr(X_k = j)$. This vector of probabilities evolves according to the equation

$$p[k] = p[k-1]\,\mathbf{P}.$$

Thus, $p[0]$ and $\mathbf{P}$ capture all the relevant information about the dynamics of the Markov chain.

The following questions are important in the study of Markov chains:

- Does there exist a π so that $\pi = \pi\mathbf{P}$? If such a π exists, it is called a stationary distribution.
- If there exists a unique stationary distribution, does $\lim_{k\to\infty} p[k] = \pi$ for all $p[0]$? In other words, does the distribution of the Markov chain converge to the stationary distribution starting from any initial state?

While the existence of a unique stationary distribution is desirable in the applications studied in this book, not all Markov chains have a unique steady-state distribution. We will first present an example of Markov chain which does not have a stationary distribution and then another example

where a stationary distribution exists but the probability distribution over the states does not converge to the stationary distribution in steady-state. Motivated by these examples, we will impose some conditions to guarantee the existence of a stationary distribution to which the Markov chain converges in steady state.

Example 3 *Consider a trivial two-state Markov chain with states a and b such that the chain remains in the initial state for time slots $k \geq 0$. Thus, the transition probability matrix for this Markov chain is given by $\mathbf{P} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Therefore, $\pi\mathbf{P} = \pi$ is true for any distribution π and the stationary distribution is not unique. The Markov chain in this example is such that if it started in one state, then it remained in the same state forever. In general, Markov chains where one state cannot be reached from some other state will not possess a unique stationary distribution.* □

The above example motivates the following definitions.

Definition 2.3.1 *Let $P_{ij}^n = \Pr(X_{k+n} = j \mid X_k = i)$.*

1. *State j is said to be* reachable *from state i, if there exists $n \geq 1$ so that $P_{ij}^n > 0$.*
2. *A Markov chain is said to be* irreducible *if a state i is reachable from any other state j.*

□

In this chapter, we will mostly consider Markov chains that are irreducible. The Markov chain in Example 3 was not irreducible.

Example 4 *Again let us consider a two-state Markov chain with two states a and b. The Markov chain behaves as follows: if it is in state a at the current time slot, then it jumps to b at the next time slot, and vice versa. Thus,*

$$\mathbf{P} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

The stationary distribution is obtained by solving $\pi\mathbf{P} = \pi$, which gives $\pi = (1/2 \quad 1/2)$. However, the system does not converge to this stationary distribution starting from any initial condition. To see this, note that if $p[0] = (1 \quad 0)$, then $p[1] = (0 \quad 1)$, $p[2] = (1 \quad 0)$, $p[3] = (0 \quad 1)$, $p[4] = (1 \quad 0)$, ... Therefore $\lim_{k\to\infty} p[k] \neq \pi$. The reason that this Markov chain does not converge to the stationary distribution is due to the fact that the state periodically alternated between a and b. □

Motivated by the above example, the following definitions lead up to the classification of a Markov chain as being either periodic or aperiodic.

Definition 2.3.2 *The following definitions classify Markov chains and their states as periodic or aperiodic:*

1. *State i is said to have a* period $d_i \geq 1$ *if d_i is the greatest integer such that $P_{ii}^n = 0$ if n is not a multiple of d_i. If $P_{ii}^n = 0$ $\forall n$, we say that $d_i = \infty$.*

2. State i is said to be aperiodic *if $d_i = 1$.*

3. A Markov chain is said to be aperiodic *if all states are aperiodic.*

□

Next, we state the following useful lemma which will be useful later to identify whether a Markov chain is aperiodic or not.

Lemma 2.3.1 *Every state in an irreducible Markov chain has the same period. Thus, in an irreducible Markov chain, if one state is aperiodic, then the Markov chain is aperiodic.* □

The Markov chain in Example 4 was not aperiodic. The following theorem states that Markov chains which do not exhibit the type of behavior illustrated in the examples possess a stationary distribution to which the distribution converges, starting from any initial state.

Theorem 2.3.2 *A finite state space, irreducible Markov chain has a unique stationary distribution π and if it is aperiodic, $\lim_{k\to\infty} p[k] = \pi$, $\forall p[0]$.* □

Example 5 *The following example illustrates the computation of the stationary distribution of a Markov chain. Consider a three-state Markov chain with the state space $\{a, b, c\}$ as shown in Figure 2.3. If the Markov chain is in state a, it switches from the current state to one of the other two*

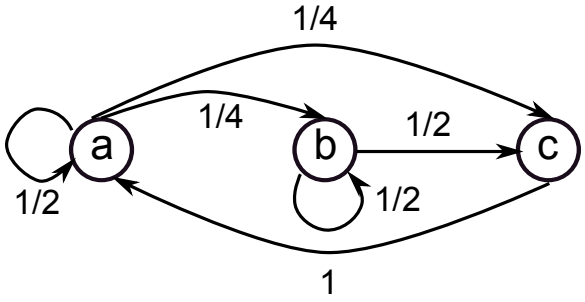

Figure 2.3: A three-state Markov chain

states, each with probability $1/4$, or remains in the same state. If it is in state b, then it switches to state c with probability $1/2$ or remains the same state. If it is in state c, it switches to state a with probability 1. Thus,

$$\mathbf{P} = \begin{pmatrix} 1/2 & 1/4 & 1/4 \\ 0 & 1/2 & 1/2 \\ 1 & 0 & 0 \end{pmatrix}.$$

This Markov chain is irreducible since it can go from any state to any other state in finite time with non-zero probability. Next, note that there is a non-zero probability of remaining in state a if the Markov chain starts in state a. Therefore, $P^n_{aa} > 0$ for all n and state a is aperiodic. Since the Markov chain is irreducible, this implies that all the states are aperiodic. Thus, the finite-state Markov chain is irreducible and aperiodic, which implies the existence of a stationary distribution

to which the probability distribution converges, starting from any initial distribution. To compute the stationary distribution π, we solve the equation

$$\pi = \pi\mathbf{P},$$

where $\pi = (\pi_a \quad \pi_b \quad \pi_c)$, *subject to the constraint* $\pi_a + \pi_b + \pi_c = 1$ *and* $\pi_a, \pi_b, \pi_c \geq 0$, *to obtain* $\pi = (1/2 \quad 1/4 \quad 1/4)$. □

If the state space is infinite, the existence of a stationary distribution is not guaranteed even if the Markov chain is irreducible as the following example illustrates.

Example 6 *Let the state space be the set of integers, and define the Markov chain as follows:*

$$\begin{aligned} X_{k+1} &= X_k + 1 \ \ w.\ p.\ 1/3, \\ &= X_k - 1 \ \ w.\ p.\ 1/3, \\ &= X_k \qquad w.\ p.\ 1/3. \end{aligned}$$

It is easy to verify that this Markov chain is irreducible and aperiodic with probability transition matrix:

$$\mathbf{P} = \begin{pmatrix} & & & \cdots & & & \\ 0 & 1/3 & 1/3 & 1/3 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 & 0 \\ & & & \cdots & & & \end{pmatrix}.$$

If a stationary distribution π exists, it has to satisfy $\pi = \pi\mathbf{P}$, which can be rewritten as

$$\pi_k = \frac{1}{3}\pi_{k-1} + \frac{1}{3}\pi_k + \frac{1}{3}\pi_{k+1}, \quad \forall k.$$

Thus, we have to solve for $\{\pi_k\}$ that satisfy

$$\begin{aligned} 2\pi_k &= \pi_{k-1} + \pi_{k+1}, \quad \forall k \\ \sum_{k=-\infty}^{\infty} \pi_k &= 1 \\ \pi_k &\geq 0, \quad \forall k. \end{aligned}$$

We will now show that we cannot find a distribution π that satisfies the above set of equations. Note that

$$\begin{aligned} \pi_2 &= 2\pi_1 - \pi_0 \\ \pi_3 &= 2\pi_2 - \pi_1 = 2(2\pi_1 - \pi_0) - \pi_1 \\ &= 3\pi_1 - 2\pi_0 \\ \pi_4 &= 6\pi_1 - 4\pi_0 - 2\pi_1 + \pi_0 \\ &= 4\pi_1 - 3\pi_0 \\ &\vdots \\ \pi_k &= k\pi_1 - (k-1)\pi_0 \\ &= (k-1)(\pi_1 - \pi_0) + \pi_1, \ \ k \geq 2 \end{aligned}$$

Thus,

$$\begin{aligned} &\text{if } \pi_1 = \pi_0 > 0, && \text{then } \pi_k = \pi_1, \ \ \forall k \geq 2 \ \text{ and } \sum_{k=0}^{\infty} \pi_k > 1, \\ &\text{if } \pi_1 > \pi_0, && \text{then } \pi_k \to \infty, \\ &\text{if } \pi_1 < \pi_0, && \text{then } \pi_k \to -\infty, \\ &\text{if } \pi_1 = \pi_0 = 0, && \text{then } \pi_k = 0 \quad \forall k \geq 0. \end{aligned}$$

A little thought shows that the last statement is also true for $k < 0$. Thus, a stationary distribution cannot exist. □

Example 6 illustrates the need for more conditions beyond irreducibility to ensure the existence of stationary distributions in countable state space Markov chains. Towards this end, we introduce the notion of recurrence and related concepts.

Definition 2.3.3 *The following definitions classify the states of a Markov chain as recurrent or transient:*

1. *The recurrence time T_i of state i of a Markov chain is defined as*

$$T_i = \min\{n \geq 1 : X_n = i \text{ given } X_0 = i.\}$$

   (Note that T_i is a random variable.)

2. *A state i is said to be* recurrent *if* $\Pr(T_i < \infty) = 1$. *Otherwise, it is called* transient.

3. *The* mean recurrence time *M_i of state i is defined as $M_i = E[T_i]$.*

4. *A recurrent state i is called* positive recurrent *if $M_i < \infty$. Otherwise, it is called* null recurrent.

5. *A Markov chain is called* positive recurrent *if all of its states are positive recurrent.*

□

The next two lemmas and theorem are stated without proof.

Lemma 2.3.3 *Suppose $\{X_k\}$ is irreducible and one of its states is positive recurrent, then all of its states are positive recurrent. (The same statement holds if we replace positive recurrent by null recurrent or transient.)* □

Lemma 2.3.4 *If state i of a Markov chain is aperiodic, then $\lim_{k\to\infty} p_i[k] = \frac{1}{M_i}$. (This is true whether or not $M_i < \infty$ and even for transient states by defining $M_i = \infty$ when state i is transient.)* □

Theorem 2.3.5 *Consider a time-homogeneous Markov chain which is irreducible and aperiodic. Then, the following results hold:*

- *If the Markov chain is positive recurrent, then there exists a unique π such that $\pi = \pi\mathbf{P}$ and $\lim_{k\to\infty} p[k] = \pi$. Further $\pi_i = \frac{1}{M_i}$.*

- *If there exists a positive vector* π *such* $\pi = \pi\mathbf{P}$ *and* $\sum_i \pi_i = 1$*, then it must be the stationary distribution and* $\lim_{k\to\infty} p[k] = \pi$*. (From Lemma 2.3.4, this also means that the Markov chain is positive recurrent.)*
- *If there exists a positive vector* π *such that* $\pi = \pi\mathbf{P}$ *and* $\sum_i \pi_i$ *is infinite, then a stationary distribution does not exist and* $\lim_{k\to\infty} p_i[k] = 0$ *for all* i*.*

□

The following example is an illustration of the application of the above theorem.

Example 7 *Consider a simple model of a wireless link where, due to channel conditions, either one packet or no packet can be served in each time slot. Let* $s(k)$ *denote the number of packets served in time slot* k *and suppose that* $s(k)$ *are i.i.d. Bernoulli random variables with mean* μ*. Further, suppose that packets arrive to this wireless link according to a Bernoulli process with mean* λ*, i.e.,* $a(k)$ *is Bernoulli with mean* λ *where* $a(k)$ *is the number of arrivals in time slot* k *and* $a(k)$ *are i.i.d. across time slots. Assume that* $a(k)$ *and* $s(k)$ *are independent processes. We specify the following order in which events occur in each time slot:*

- *We assume that any packet arrival occurs first in the time slot, followed by any packet departure.*
- *Packets that are not served in a time slot are queued in a buffer for service in a future time slot.*

Let $q(k)$ *be the number of packets in the queue at the beginning of time slot* k*. Then* $q(k)$ *is a Markov chain and evolves according to the equation*

$$q(k+1) = (q(k) + a(k) - s(k))^+ .$$

We are interested in the steady state distribution of this Markov chain. The Markov chain can be pictorially depicted as in Figure 2.4 where the circles denote the states of the Markov chain (the number of packets in the queue) and the arcs denote the possible transitions with the number on an arc denoting the probability of that transition occurring. For example, $P_{i,i+1}$*, the probability that*

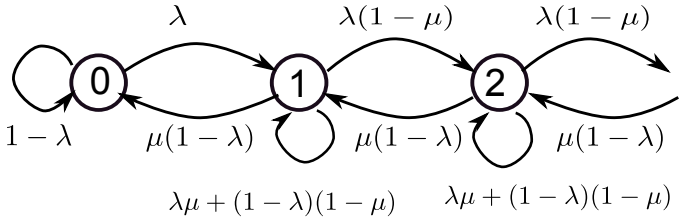

Figure 2.4: The discrete-time Markov chain for the queue

the number of packets in the queue increases from i *to* $i+1$ *from one time slot to the next is equal to the probability that there was an arrival but no departure in the time slot. Thus,*

$$P_{i,i+1} = \lambda(1-\mu).$$

Similarly, P_{ii} for $i > 0$ is equal to the probability of no arrival and no departure or one arrival and one departure in a time slot and thus,

$$P_{ii} = \lambda\mu + (1-\lambda)(1-\mu).$$

On the other hand, P_{00} is simply the probability of no arrival which is equal to $1-\lambda$. If $i > 0$, then $P_{i,i-1} = 1 - P_{ii} - P_{i,i+1}$. First,we note that it is easy to see that this Markov chain is irreducible and aperiodic. To compute π, we have to solve for $\pi = \pi\mathbf{P}$ which can be written explicitly as

$$\begin{aligned} \pi_i &= \pi_{i-1}P_{i-1,i} + \pi_i P_{ii} + \pi_{i+1}P_{i+1,i}, \qquad i > 0 \\ \pi_0 &= \pi_0 P_{00} + \pi_1 P_{10}. \end{aligned} \tag{2.10}$$

The above equations have a simple interpretation: the stationary probability of being in i is equal to sum of the probability of being in state i in the previous time-slot multiplied by the probability of continuing in the same state and the probability being in another state and making a transition to state i. The above set of equations should be augmented with the constraint $\sum_i \pi_i = 1$ to solve for π.

Using the fact $P_{ii} + P_{i,i-1} + P_{i,i+1} = 1$, a little thought shows that if we find π that satisfies

$$\pi_i P_{i,i+1} = \pi_{i+1}P_{i+1,i}, \quad \forall i,$$

then it also solves (2.10). Thus,

$$\pi_{i+1} = \frac{(1-\mu)\lambda}{(1-\lambda)\mu}\pi_i,$$

which implies

$$\pi_i = \left(\frac{(1-\mu)\lambda}{(1-\lambda)\mu}\right)^i \pi_0. \tag{2.11}$$

Since $\sum_{i\geq 0}\pi_i = 1$, we obtain

$$\pi_0 \sum_{i=0}^{\infty}\left(\frac{(1-\mu)\lambda}{(1-\lambda)\mu}\right)^i = 1.$$

If we assume $\lambda < \mu$, then

$$\frac{(1-\mu)\lambda}{(1-\lambda)\mu} < 1,$$

and

$$\pi_0 = 1 - \frac{(1-\mu)\lambda}{(1-\lambda)\mu}.$$

Thus, the stationary distribution is completely characterized.

If $\lambda \geq \mu$, then let $\pi_0 = 1$ and from (2.11), it is clear that $\sum_i \pi_i = \infty$. Thus, from Theorem 2.3.5, the Markov chain is not positive recurrent if $\lambda \geq \mu$. □

As noted in the above example, the following theorem provides a sufficient condition for verifying the stationary distribution of a DTMC.

Theorem 2.3.6 *Consider a time-homogeneous Markov chain which is irreducible and aperiodic. If there exists a positive vector π such $\pi_i P_{ij} = \pi_j P_{ji}$ and $\sum_i \pi_i = 1$, then it must be the stationary distribution.* □

Unlike the above example, there are also many instances where one cannot easily find the stationary distribution by solving $\pi = \pi\mathbf{P}$. But we would still like to know if the stationary distribution exists. It is often easy to verify the irreducibility and aperiodicity of a Markov chain, but, in general, it is difficult to directly verify whether a Markov chain is positive recurrent from the definitions given earlier. Instead, there is a convenient test called Foster's test or the Foster-Lyapunov test to check for positive recurrence which we state next.

Theorem 2.3.7 (Foster-Lyapunov Theorem) *Let $\{X_k\}$ be an irreducible Markov chain with a state space $\mathcal{S}$. Suppose that there exist a function $V : \mathcal{S} \to \mathcal{R}^+$ and a finite set $\mathcal{B} \subseteq \mathcal{S}$ satisfying the following conditions:*

1. *$E[V(X_{k+1}) - V(x) \mid X_k = x] \leq -\epsilon$ if $x \in \mathcal{B}^c$ for some $\epsilon > 0$, and*
2. *$E[V(X_{k+1}) - V(x) \mid X_k = x] \leq A$ if $x \in \mathcal{B}$ for some $A < \infty$.*

Then the Markov chain $\{X_k\}$ is positive recurrent.

Proof We will prove the result under the further assumption that $\{X_k\}$ is aperiodic. Note that the theorem itself does not require the Markov chain to be aperiodic. Following the two conditions stated in the theorem, we have

$$\begin{aligned} E[V(X_{k+1}) - V(X_k) \mid X_k = x] &\leq -\epsilon \mathbb{I}_{x\in\mathcal{B}^c} + A\mathbb{I}_{x\in\mathcal{B}} \\ &= -\epsilon \mathbb{I}_{x\in\mathcal{B}^c} + A - A\mathbb{I}_{x\in\mathcal{B}^c} \\ &= -(A+\epsilon)\mathbb{I}_{x\in\mathcal{B}^c} + A. \end{aligned}$$

Taking expectations on both sides, we get

$$\begin{aligned} E[V(X_{k+1})] - E[V(X_k)] &= -(A+\epsilon)\Pr(X_k \in \mathcal{B}^c) + A, \\ \sum_{k=0}^{N} (E[V(X_{k+1})] - E[V(X_k)]) &= -(A+\epsilon)\sum_{k=0}^{N} \Pr(X_k \in \mathcal{B}^c) + AN. \end{aligned}$$

Thus,

$$\begin{aligned} E[V(X_{N+1})] - E[V(X_0)] &= -(A+\epsilon)\sum_{k=0}^{N} \Pr(X_k \in \mathcal{B}^c) + AN \\ (A+\epsilon)\sum_{k=0}^{N} \Pr(X_k \in \mathcal{B}^c) &= AN + E[V(X_0)] - E[V(X_{N+1})] \\ &\leq AN + E[V(X_0)] \\ \frac{A+\epsilon}{N}\sum_{k=0}^{N} \Pr(X_k \in \mathcal{B}^c) &\leq A + \frac{1}{N}E[V(X_0)] \\ \lim_{N\to\infty} \sup \frac{1}{N}\sum_{k=0}^{N} \Pr(X_k \in \mathcal{B}^c) &\leq \frac{A}{A+\epsilon} \\ \lim_{N\to\infty} \inf \frac{1}{N}\sum_{k=0}^{N} \Pr(X_k \in \mathcal{B}) &\geq \frac{\epsilon}{A+\epsilon} \end{aligned}$$

Suppose that any state of the Markov chain is not positive recurrent, all states are not positive recurrent since the Markov chain is irreducible. Thus, $M_i = \infty$, $\forall i$ and $\lim_{k\to\infty} p_i[k] = 0$, $\forall i$. Thus, $\lim_{k\to\infty} \Pr(X_k \in \mathcal{B}) = 0$ or $\lim_{N\to\infty} \inf \frac{1}{N} \sum_{k=1}^{N} \Pr(X_k \in \mathcal{B}) = 0$, which contradicts the fact that it is $\geq \frac{\epsilon}{A+\epsilon}$.

□

Next, we present two extensions of the Foster-Lyapunov Theorem without proof.

Theorem 2.3.8 *An irreducible Markov chain $\{X_k\}$ is positive recurrent if there exists a function $V : \mathcal{S} \to \mathcal{R}^+$, a positive integer $L \geq 1$ and a finite set $\mathcal{B} \subseteq \mathcal{S}$ satisfying the following conditions:*

$$E[V(X_{k+L}) - V(x) \mid X_k = x] \leq -\epsilon \mathbb{I}_{x\in\mathcal{B}^c} + A\mathbb{I}_{x\in\mathcal{B}}$$

for some $\epsilon > 0$ and $A < \infty$. . □

Theorem 2.3.9 *An irreducible Markov chain $\{X_k\}$ is positive recurrent if there exists a function $V : \mathcal{S} \to \mathcal{R}^+$, a function $\eta : \mathcal{S} \to \mathcal{R}^+$, and a finite set $\mathcal{B} \subseteq \mathcal{S}$ satisfying the following conditions:*

$$E[V(X_{k+\eta(x)}) - V(x) \mid X_k = x] \leq -\epsilon\eta(x) \mathbb{I}_{x\in\mathcal{B}^c} + A\mathbb{I}_{x\in\mathcal{B}}$$

for some $\epsilon > 0$ and $A < \infty$. □

The following theorem provides conditions under which a Markov chain is not positive recurrent.

Theorem 2.3.10 *An irreducible Markov chain $\{X_k\}$ is either transient or null recurrent if there exists a function $V : \mathcal{S} \to \mathcal{R}^+$ and and a finite set $\mathcal{B} \subseteq \mathcal{S}$ satisfying the following conditions:*

- $E\left[V(X_{k+1}) - V(X_k) \mid X_k = x\right] \geq 0, \quad \forall x \in \mathcal{B}^c$
- *There exists some $x \in \mathcal{B}^c$ such that $V(x) > V(y)$ for all $y \in \mathcal{B}$, and*
- $E\left[|V(X_{k+1}) - V(X_k)| \,|\, X_k = x\right] \leq A$ *for some $A < \infty$ and $\forall x \in \mathcal{S}$.*

□

2.4 Delay and Packet Loss Analysis in Queues

2.4.1 Little's Law

We start this section with the famous Little's law, which states that the expected waiting time in queueing system, is equal to the product of the mean arrival rate and the mean queue length. Little's law holds for very general arrival processes and service disciplines, and for both discrete time and continuous time queueing systems. In this book, we only derive Little's law for discrete time queueing systems. The derivation for continuous time systems is similar.

We assume packets arrive at the beginning of a time slot and are served at the end of a time slot. The queue length at time slot t, denoted by $q(t)$, is the number of packets remaining in the system at the beginning of time slot t, *before packet arrivals occur.*

Let $A(t)$ denote the number of packet arrivals up to but *not including* time slot t, and $\mathbb{I}_i(t)$ be an indicator of the presence of packet i in the queue at time t, i.e.,

$$\mathbb{I}_i(t) = \begin{cases} 1, & \text{if packet } i \text{ arrived in a time slot } < t \text{ and departed in a time slot } \geq t, \\ 0, & \text{otherwise.} \end{cases},$$

where packets are indexed according to arrival times and ties are broken arbitrarily. Note that $\mathbb{I}_i(t) = 0$ if packet i arrives in time slot t. Since $\mathbf{I}_i(t) = 1$ means packet i remains in the system at the beginning of time slot t, $q(t)$ can be written as

$$q(t) = \sum_{i=1}^{A(t)} \mathbb{I}_i(t).$$

Further, the waiting time of packet i, denoted by w_i, is defined to be

$$w_i = \sum_{t=1}^{\infty} \mathbb{I}_i(t).$$

Note that according this definition, the waiting time of a packet is zero if the packet arrives and departs in the same time slot.

We define $\lambda(T)$ to the average arrival rate by time slot T, i.e.,

$$\lambda(T) = \frac{A(T+1)}{T};$$

and $L(T)$ to be the average queue length by time slot T, i.e.,

$$L(T) = \frac{\sum_{t=1}^{T} q(t)}{T}.$$

Further, we define $W(n)$ to be the average waiting time of the first n packets that departed from the system, i.e.,

$$W(n) = \frac{1}{n} \sum_{k=1}^{n} w_{i_k},$$

where i_k is the index of the k^{th} packet that left the system. We further define the following three limits:

$$\lambda = \lim_{T \to \infty} \lambda(T), \quad L = \lim_{T \to \infty} L(T) \quad \text{and} \quad W = \lim_{n \to \infty} W(n).$$

So λ is the average arrival rate, L is the average queue length, and W is the average waiting time.

Theorem 2.4.1 (Little's Law) *Assuming that λ and W exist and are finite, L exists and $L = \lambda W$.*

Proof According to the definition of $L(T)$, we have

$$L(T) = \frac{1}{T}\sum_{t=1}^{T} q(t) = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{A(t)} \mathbb{I}_i(t) = \frac{1}{T}\sum_{i=1}^{A(T)}\sum_{t=1}^{T} \mathbb{I}_i(t).$$

We first derive an upper bound on L. Note that

$$\sum_{t=1}^{T} \mathbb{I}_i(t) \le \sum_{t=1}^{\infty} \mathbb{I}_i(t) = w_i,$$

so

$$L(T) = \frac{1}{T}\sum_{i=1}^{A(T)}\sum_{t=1}^{T} \mathbb{I}_i(t) \le \frac{1}{T}\sum_{i=1}^{A(T)} w_i.$$

When λ and W exist and are finite, we have

$$\lim_{T\to\infty} \frac{1}{T}\sum_{i=1}^{A(T)} w_i = \lim_{T\to\infty} \frac{T-1}{T}\frac{A(T)}{T-1}\frac{\sum_{i=1}^{A(T)} w_i}{A(T)} = \lambda W,$$

so

$$L = \lim_{T\to\infty} L(T) \le \lambda W. \tag{2.12}$$

Next we show that λW is also a lower bound on L. To prove that, we denote $\mathcal{D}(T)$ to be the set of packets that have departed before time slot T. Note that the packets can be served in an arbitrary order, so $\mathcal{D}(T)$ can be an arbitrary subset of $\{1, \cdots, A(T)\}$.

Since any packet i ($i \in \mathcal{D}(T)$) departed from the buffer before time slot T, we have $w_i = \sum_{t=1}^{T-1} \mathbb{I}_i(t)$, which leads to the following lower bound on $L(T)$:

$$L(T) = \frac{1}{T}\sum_{i=1}^{A(T)}\sum_{t=1}^{T} \mathbb{I}_i(t) \ge \frac{1}{T}\sum_{i\in\mathcal{D}(T)} w_i = \frac{|\mathcal{D}(T)|}{T} \times \frac{1}{|\mathcal{D}(T)|}\sum_{i\in\mathcal{D}(T)} w_i.$$

Now, if the following two limits hold

$$\lim_{T\to\infty} \frac{|\mathcal{D}(T)|}{T} \ge \lambda \quad \text{and} \lim_{T\to\infty} \sum_{i\in\mathcal{D}(T)} w_i \ge W, \tag{2.13}$$

then we have $L \ge \lambda W$, so the theorem holds.

To prove (2.13), we will establish the following claim: Given any $\epsilon > 0$, there exist constants a_ϵ, b_ϵ and δ_ϵ such that (i) $\delta_\epsilon \to 0$ as $\epsilon \to 0$, and (ii) for any $T \ge b_\epsilon$,

$$\{a_\epsilon, \cdots, A((1-\delta_\epsilon)T)\} \subseteq \mathcal{D}(T).$$

This claim indicates for sufficiently large T, most packet that arrived before time slot T must have departed by time slot T. From this claim, it is straightforward to show (2.13).

We now verify the claim to complete the proof. Since $\lim_{n\to\infty} W(n)/n$ exists and is finite,

$$\begin{aligned}\lim_{n\to\infty}\frac{w_n}{n} &= \lim_{n\to\infty}\frac{\sum_{i=1}^{n} w_i}{n} - \lim_{n\to\infty}\frac{n-1}{n}\frac{\sum_{i=1}^{n-1} w_i}{n-1}\\ &= W - W\\ &= 0.\end{aligned}$$

So given any $\epsilon > 0$, there exists n_ϵ such that $w_n \leq \epsilon n$ for all $n \geq n_\epsilon$. Further, $\lim_{T\to\infty}\frac{A(T)}{T} = \lambda$ implies that given $\epsilon > 0$, there exists T_ϵ such that

$$(\lambda - \epsilon)T \leq A(T) \leq (\lambda + \epsilon)T \tag{2.14}$$

for any $T \geq T_\epsilon$.

Given ϵ, we choose T large enough such that

$$T \geq \frac{T_\epsilon}{1 - \epsilon(\lambda + \epsilon)} \tag{2.15}$$

$$T \geq \frac{n_\epsilon}{(\lambda - \epsilon)(1 - \epsilon(\lambda + \epsilon))}. \tag{2.16}$$

Since $(1 - \epsilon(\lambda + \epsilon))T \geq T_\epsilon$ according to (2.15),

$$A((1 - \epsilon(\lambda + \epsilon))T) \geq (\lambda - \epsilon)(1 - \epsilon(\lambda + \epsilon))T \geq n_\epsilon,$$

where the first and second inequalities follows from condition (2.14) and (2.16), respectively. Now for any packet i such that $n_\epsilon \leq i \leq A((1 - \epsilon(\lambda + \epsilon))T)$, according to the definition of n_ϵ,

$$w_i \leq \epsilon i \leq \epsilon A((1 - \epsilon(\lambda + \epsilon))T) \leq \epsilon(\lambda + \epsilon)(1 - \epsilon(\lambda + \epsilon))T \leq \epsilon(\lambda + \epsilon)T,$$

where the third inequality holds due to conditions (2.14) and (2.15). To that end, packet i for $n_\epsilon \leq i \leq A((1 - \epsilon(\lambda + \epsilon))T)$ must depart by

$$(1 - \epsilon(\lambda + \epsilon))T - 1 + w_i \leq (1 - \epsilon(\lambda + \epsilon))T - 1 + \epsilon(\lambda + \epsilon)T = T - 1,$$

so

$$\{n_\epsilon, \cdots, A((1 - \epsilon(\lambda + \epsilon))T\} \subseteq \mathcal{D}(T).$$

The claim therefore holds with $a_\epsilon = n_\epsilon$, $b_\epsilon = \max\left\{\frac{T_\epsilon}{1-\epsilon(\lambda+\epsilon)}, \frac{n_\epsilon}{(\lambda-\epsilon)(1-\epsilon(\lambda+\epsilon))}\right\}$, and $\delta_\epsilon = \epsilon(\lambda + \epsilon)$ which goes to 0 as $\epsilon \to 0$.

□

The above derivation of Little's law assumes that L, λ, and W are sample path averages. In applications, we will apply Little's law to steady-state expected queue lengths, steady-state expected arrival rates and steady-state expected waiting times. Thus, we will make an implicit assumption throughout the book that the stochastic processes that we consider are ergodic, i.e., processes for which steady-state expectations and sample-path averages are equal. Further, we will also assume that λ and W exist as required by Little's law.

Next we will consider single server queueing systems under various traffic and buffer models. The results help us understand queueing delays and packet drop probabilities over a link.

2.4.2 The Geo/Geo/1 Queue

We consider a single-server queue with infinite buffer space. Packets arrive to this queue according to an i.i.d. Bernoulli process with parameter λ. In each time slot, either one packet is served with probability μ or no packet is served with probability $1-\mu$. Equivalently, we can assume that the server serves one unit of data per time slot and packet sizes are geometrically distributed with mean $1/\mu$. Under these assumptions, the inter-arrival and departure times are geometrically distributed, so the queue is called Geo/Geo/1 queue.

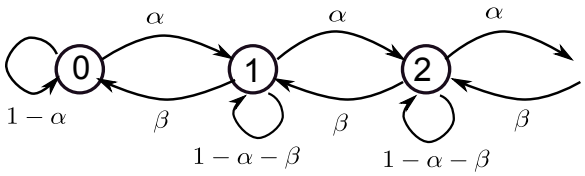

Figure 2.5: The birth-death process of Geo/Geo/1 queue

This simple queuing system can be viewed as a birth-death process as shown in Figure 2.5, where the state of the Markov chain is the queue length. The parameter

$$\alpha = \Pr(\text{1 arrival, no departure}) = \lambda(1-\mu)$$

is the probability that the queue length increases by one, and

$$\beta = \Pr(\text{no arrival, 1 departure}) = (1-\lambda)\mu$$

is the probability that the queue length decreases by one.

Let π denote the steady-state distribution of the Markov chain. We will attempt to find π by solving the local balance equation

$$\beta\pi_{i+1} = \alpha\pi_i,$$

Dividing β at both sides yields

$$\pi_{i+1} = \rho\pi_i \tag{2.17}$$

where $\rho = \frac{\alpha}{\beta} = \frac{\lambda(1-\mu)}{\mu(1-\lambda)}$. Since equality (2.17) holds for all i, we can further obtain

$$\pi_i = \rho^i\pi_0. \tag{2.18}$$

Normalizing to make the sum of the probabilities equal to one, we have

$$\sum_{i=0}^{\infty}\pi_i = \pi_0\sum_{i=0}^{\infty}\rho^i = 1. \tag{2.19}$$

If $\rho < 1$, i.e., $\frac{\lambda}{\mu} < 1$, then from (2.19), we get $\frac{\pi_0}{1-\rho} = 1$, and $\pi_0 = 1-\rho$. According to (2.18),

$$\pi_i = \rho^i(1-\rho)$$

and the Markov chain is positive recurrent. If $\rho \geq 1$, then $\sum_i \pi_i = \infty$. So the Markov chain is not positive recurrent.

Assume that $\rho < 1$, then the average queue length

$$\begin{aligned} E[q] &= \sum_{i=0}^{\infty} \rho^i (1-\rho) i \\ &= (1-\rho)\rho \sum_{i=1}^{\infty} i\rho^{i-1} \\ &= (1-\rho)\rho \frac{1}{(1-\rho)^2} \\ &= \frac{\rho}{1-\rho}. \end{aligned}$$

By Little's law, the average waiting time of a packet is

$$W = \frac{L}{\lambda} = \frac{\rho}{\lambda(1-\rho)}.$$

2.4.3 The Geo/Geo/1/B Queue

We now consider the same queuing model as the previous subsection, but we further assume that the buffer size is finite. We let the maximum buffer size be denoted by B, i.e., B is the maximum number of packets allowed in the queue. When the buffer is full, newly arriving packets are dropped. It is easy to see that $q(t)$ again is a Markov chain as shown in Figure 2.6, and the steady-state distribution satisfies:

$$\beta \pi_{i+1} = \alpha \pi_i$$

for $0 \le i \le B-1$. Defining $\rho = \frac{\alpha}{\beta}$, we have

$$\pi_{i+1} = \rho \pi_i$$

for $0 \le i \le B-1$. Normalizing the probabilities, we have $\pi_0 \sum_{i=0}^{B} \rho^i = 1$, which implies that

$$\pi_0 \frac{1-\rho^{B+1}}{1-\rho} = 1.$$

Hence, we obtain

$$\begin{aligned} \pi_0 &= \frac{1-\rho}{1-\rho^{B+1}} \\ \pi_i &= \frac{(1-\rho)\rho^i}{1-\rho^{B+1}}, \quad i = 0, 1, 2, \cdots, B. \end{aligned}$$

Note that the Markov chain is positive recurrent for any ρ because this is a finite-state Markov chain. Therefore, in this Geo/Geo/1/B queue model, we should look at *the fraction of arriving packets that are dropped.*

Denote by p_d the fraction of arriving packets that are dropped, i.e.,

$$p_d = \lim_{T\to\infty} \frac{\sum_{t=0}^{T} \mathbb{I}_{\{q(t)=B, a(t)=1\}}}{\sum_{t=0}^{T} \mathbb{I}_{\{a(t)=1\}}},$$

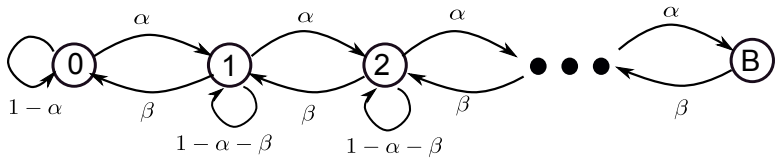

Figure 2.6: The birth-death process of Geo/Geo/1/B queue

where $a(t) = 1$ if there is a packet arrival at time t, and $a(t) = 0$ otherwise. In other words, p_d is the fraction of time slots that the buffer is full when there is an arrival.

Assuming ergodicity and $q(t)$ is in its steady-state, we obtain

$$\begin{aligned} p_d &= \lim_{T\to\infty} \frac{\frac{1}{T}\sum_{t=1}^{T} \mathbb{I}_{\{q(t)=B, a(t)=1\}}}{\frac{1}{T}\sum_{t=1}^{T} \mathbb{I}_{\{a(t)=1\}}} \\ &= \frac{\Pr(a(t)=1, q(t)=B)}{\Pr(a(t)=1)} \\ &= \Pr(q(t)=B | a(t)=1). \end{aligned}$$

Since the arrivals are i.i.d. across time slots, and independent of the services, $q(t)$ is independent of $a(t)$, which implies that

$$p_d = \Pr(q(t)=B) = \pi_B. \tag{2.20}$$

Thus, we have used the fact that the next arrival is independent of past arrivals and services. This observation may not be true for other arrivals, but it holds for Bernoulli arrival processes. Equation (2.20) is a special case of more general result called the BASTA property: Bernoulli Arrivals See Time Averages since p_d (what the arrivals see) is equal to π_B (the time average).

2.4.4 The Discrete-Time G/G/1 Queue

We now consider a G/G/1 queue where the first G refers to the fact that arrival process is general and the second G refers to the fact that the service process is general. We assume the arrivals (and potential departures) are i.i.d. across time and the arrival process is independent of the potential departure process. We also assume both the arrival process and potential departure process have finite second moments.

The queue dynamics can be described as:

$$q(t+1) = (q(t) + a(t) - s(t))^+ ,$$

where $a(t)$ is the number of packet arrivals at the beginning of time slot t, and $s(t)$ is the number of packet departures at the end of time slot t. Assume the first and second moments of $a(t)$ and $s(t)$ exist, and

$$\begin{aligned} E[a(t)] = \lambda \quad &\text{and} \quad E[a^2(t)] = m_{2a} \\ E[s(t)] = \mu \quad &\text{and} \quad E[s^2(t)] = m_{2s}. \end{aligned}$$

The next theorem states that the queuing system is positive recurrent when $\lambda < \mu$, i.e., mean arrival rate is strictly less than mean service rate.

Theorem 2.4.2 *The discrete-time G/G/1 queue is positive recurrent when $\lambda < \mu$. Further, using $q(\infty)$ to informally denote the queue in steady-state, the following inequality holds*

$$E\left[q(\infty)\right] \leq \frac{m_{2a} + m_{2s} - 2\lambda\mu}{2(\mu - \lambda)}.$$

Proof Considering the Lyapunov function

$$V(t) \triangleq \frac{1}{2}q^2(t),$$

the drift of the Lyapunov function is given by

$$\begin{aligned} E\left[V(t+1) - V(t) | q(t) = q\right] &= \frac{1}{2}E\left[((q + a(t) - s(t))^+)^2 - q^2\right] \\ &\leq \frac{1}{2}E\left[(q + a(t) - s(t))^2 - q^2\right] \\ &= \frac{1}{2}E\left[(a(t) - s(t))^2 + 2q(a(t) - s(t))\right] \\ &= \frac{1}{2}[m_{2a} + m_{2s} - 2\lambda\mu + 2q(\lambda - \mu)] \\ &= \frac{m_{2a} + m_{2s} - 2\lambda\mu}{2} + q(\lambda - \mu). \end{aligned} \tag{2.21}$$

Fix $\epsilon > 0$. If

$$\frac{m_{2a} + m_{2s} - 2\lambda\mu}{2} + q(\lambda - \mu) \leq -\epsilon$$

or in other words,

$$q \geq \left(\epsilon + \frac{m_{2a} + m_{2s} - 2\lambda\mu}{2}\right)\frac{1}{\mu - \lambda},$$

then

$$E\left[V(t+1) - V(t) | q(t) = q\right] \leq -\epsilon.$$

Invoking the Foster-Lyapunov Theorem (Theorem 2.3.7), we conclude that the Markov chain is positive recurrent, and has a stationary distribution.

We will now obtain a bound on the mean queue length. Taking expectations on both sides of inequality (2.21), we have

$$E\left[V(t+1) - V(t)\right] \leq \frac{m_{2a} + m_{2s} - 2\lambda\mu}{2} + E\left[q(t)\right](\lambda - \mu).$$

Supposing the system is in steady-state and $E[V(t)]$ exists in steady-state, we have

$$E\left[V(t+1) - V(t)\right] = 0.$$

Using $q(\infty)$ to informally denote steady-state, we get

$$E\left[q(\infty)\right] \leq \frac{m_{2a} + m_{2s} - 2\lambda\mu}{2(\mu - \lambda)},$$

which we call the *discrete-time Kingman bound.*

□

The discrete-time Kingman bound is actually tight when $\lambda \to \mu$. We define $\bar{q}(\lambda) \triangleq E[q(\infty)]$ to be the expected queue length when the arrival rate is λ.

Assume that $\lim_{\lambda\to\mu} m_{2a}(\lambda) = \hat{m}_{2a}$ exists. For example, for i.i.d. Bernoulli arrivals,

$$m_{2a} = E[a^2(t)] = \lambda,$$

so

$$\lim_{\lambda\to\mu} m_{2a} = \mu.$$

Then as $\lambda \to \mu$, the discrete-time Kingman bound becomes

$$\lim_{\lambda\to\mu} (\mu - \lambda)\bar{q}(\lambda) = \frac{\hat{m}_{2a} + m_{2s} - 2\mu^2}{2}. \tag{2.22}$$

Using Little's law, we also have an upper bound on the queueing delay given by

$$W \leq \frac{m_{2a} + m_{2s} - 2\lambda\mu}{2\lambda(\mu - \lambda)}.$$

Chapter 3

Scheduling in Packet Switches

In Chapter 1, we learned about routing algorithms that determine the sequence of links a packet should traverse to get to its destination. But we did not explain how a router actually moves a packet from one link to another. To understand this process, let us first look at the architecture of a router. Generally speaking, a router has four major components: the input and the output ports which are interfaces connecting the router to input and output links, respectively, a switch fabric and a routing processor, as shown in Figure 3.1. The routing processor maintains the routing table and makes routing decisions. *The switch fabric is the component that moves packets from one link to another link.*

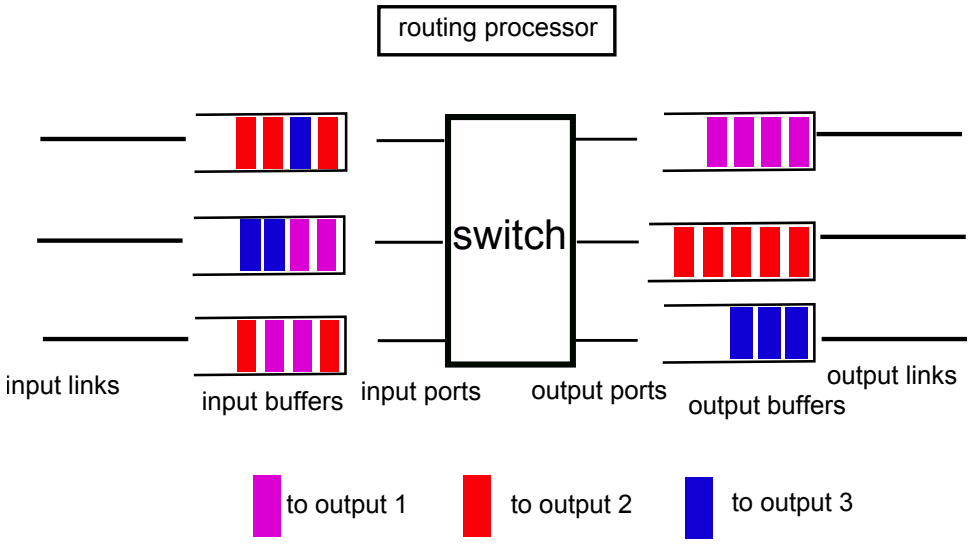

Figure 3.1: A router has four major components: input ports, output ports, switch fabric and routing processor. Each input/output port is connected to an input/output link and maintains an input/output buffer. Packets are moved from input buffers to output buffers via the switch fabric.

Earlier, we *implicitly assumed* that this switch fabric operates infinitely fast, so packets are moved from input ports to output ports immediately. This allowed us to focus on the buffers at output ports. So all our discussions so far on buffer overflow probabilities are for output queues

since an output buffer is the place where packets "enter" a link. However, in reality, the switch fabric does not really operate at infinite speed. In particular, packets encounter queueing delays at input ports before they are transferred to output ports. Therefore, buffers are required at both input and output ports to store packets. The question we now address is *how to operate the switch fabric so that the average queue lengths at input port buffers are finite?*

3.1 Switch Architectures and Crossbar Switches

The most popular switch architecture today is the crossbar switch architecture. Figure 3.2 shows a crossbar switch with three input ports and three output ports. An $N \times N$ crossbar switch (i.e., a switch with N input ports and N output ports) consists of N^2 crosspoints. An input port is connected to an output port when the corresponding crosspoint is closed as shown in Figure 3.2. At most one crosspoint can be closed at each row and at most one can be closed at each column. In other words, at any given time, one input port can be connected to at most one output port, and vice versa.

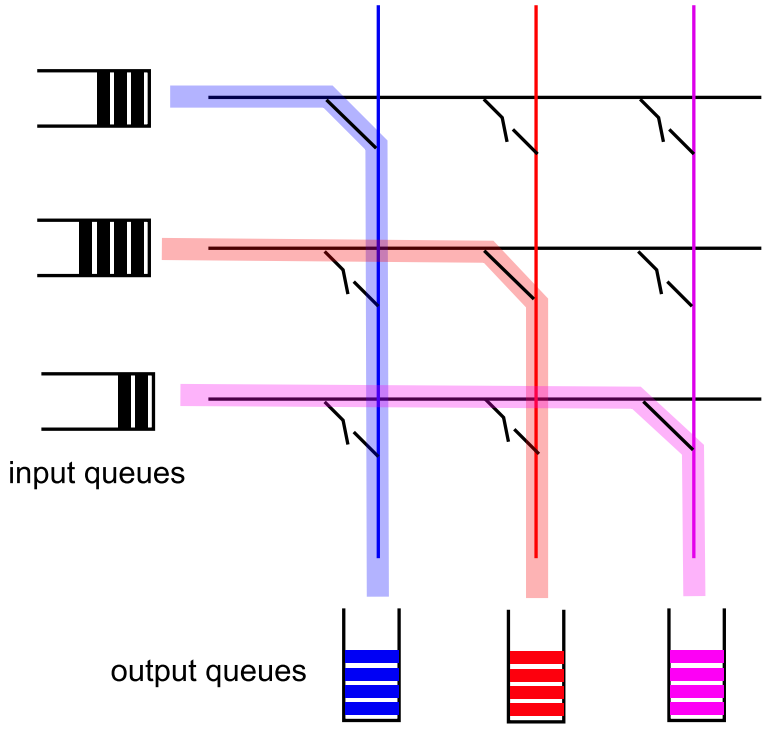

Figure 3.2: A crossbar switch with three input ports and three output ports. Three crosspoints are closed, connecting three input ports to three output ports.

A simple way to represent a crossbar switch is to use a complete bipartite graph as shown in Figure 3.3. An $N \times N$ complete bipartite graph consists of N^2 edges, representing the N^2 crosspoints. The definitions of bipartite and complete bipartite graphs are presented below.

Definition 3.1.1 (Bipartite Graph) *A* bipartite graph *is a graph whose vertices can be parti-*

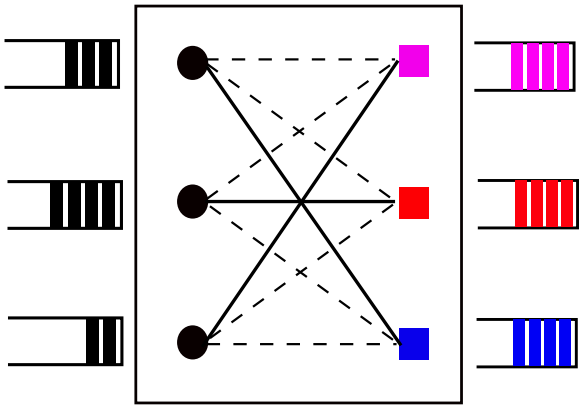

Figure 3.3: The bipartite graph representation of the switch in Figure 3.2

tioned into two sets $\mathcal{I}$ and $\mathcal{O}$ such that all edges connect a vertex in $\mathcal{I}$ to a vertex in $\mathcal{O}$. For a crossbar switch, $\mathcal{I}$ is the set of input ports and $\mathcal{O}$ is the set of output ports. □

Definition 3.1.2 (Complete Bipartite Graph) *A* complete *bipartite graph is a bipartite graph where every vertex in $\mathcal{I}$ is connected to every vertex in $\mathcal{O}$.* □

Definition 3.1.3 (Matching) *In a graph, a matching is a set of edges such that no two share a common vertex. In other words, each node is associated with at most one edge in a matching. Figure 3.4 illustrates both a valid matching and an invalid matching.* □

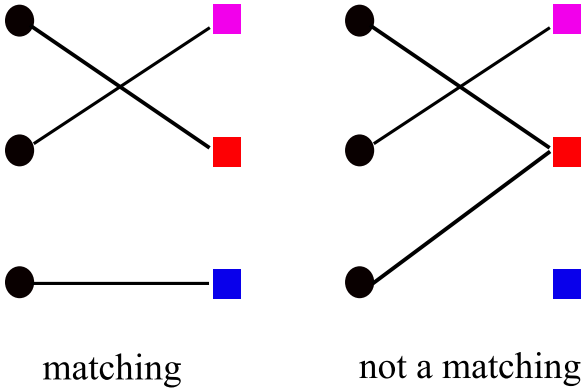

Figure 3.4: Matching in a graph

Definition 3.1.4 (Schedule) *In a crossbar switch, a schedule is a set of connections from input ports to output ports such that no input port is connected to two output ports and vice versa.* □

According to the definitions of a matching and a schedule, a schedule has to be a valid matching in the bipartite graph representing the switch. An important problem in crossbar switch design is

to develop scheduling algorithms that find *good* matchings at each time instant to transfer packets from input buffers to output buffers, which is the main focus of this chapter. We assume that all packets are of equal size. If this is not the case, then packets are divided into equal sized chunks called cells at input ports and reassembled at the output ports. We do not consider this fragmentation and reassembly process in this book.

3.1.1 Head-of-Line (HOL) Blocking and Virtual Output Queues

When the speed of a switch is not fast enough, packets will be queued at input buffers. We first consider the case where each input port is associated with a queue where packets are transmitted in a First-In, First-Out fashion, called a FIFO queue, the switch operates in a time-slotted fashion, and one and at most one packet can be transferred from an input queue to an output queue during one time slot if they are connected. Assume the status of input queues at the beginning of a time slot is as shown in Figure 3.5. If the switch can freely select any packet in an input queue to transfer, then the switch can transfer one packet from input queue 1 to output queue 2, one from input queue 2 to output queue 3, and one from input queue 3 to output queue 1. However, because input queues are FIFO queues, the switch can only access the first packet in each queue (also known as the head-of-the-line packet). Since no head-of-line packet is destined to output port 1, even though output port 1 is free, those packets destined to output port 1 cannot be transferred. Therefore, the switch can make only two transfers as shown in Figure 3.5. This phenomena whereby packets behind the first packet in each queue may be blocke is called *Head-of-Line (HOL) blocking*, which can significantly degrade the performance of a switch.

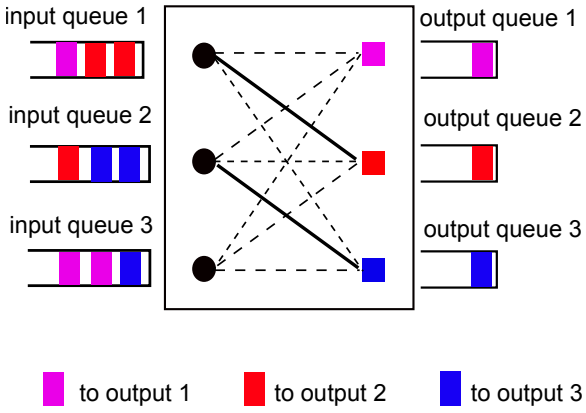

Figure 3.5: HOL blocking at input queues

HOL blocking occurs because the switch can only access head-of-line packets. So each input queue can only be connected to the output queue to which its head-of-line packet is destined, which significantly limits the flexibility in selecting a schedule. One approach to resolve this HOL blocking problem is to maintain a separate queue for each output port at each input port, called *virtual output queues (VOQs),* as shown in Figure 3.6. With VOQs, at each input port, the switch can access one packet for each output port (if such a packet exists). Assuming all VOQs are nonempty, an input port can transfer one packet to any output port. Consider the same situation as in Figure 3.5, but with VOQs, the switch can transfer one packet to each output queue simultaneously as shown in

Figure 3.6, so is fully utilized.

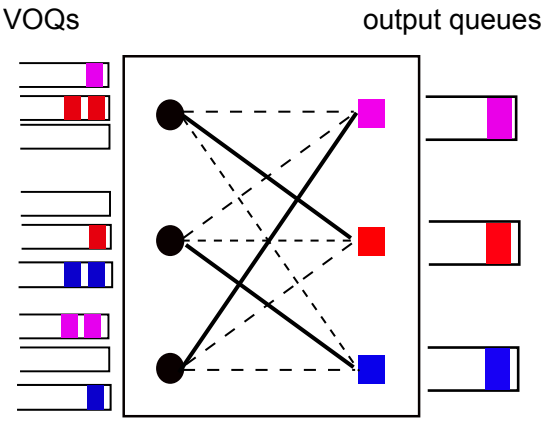

Figure 3.6: HOL blocking at input queues

3.2 Capacity Region and MaxWeight Scheduling

While VOQs resolve the HOL blocking problem, it is yet clear whether the switch can operate at its full capacity To address this question, we first need to understand what "full capacity" means, i.e., understand the capacity region of a switch. We let VOQ(i,j) denote the VOQ for output port j at input port i, and $q_{ij}(t)$ denote the length of VOQ(i,j) at time slot t. Further, denote by $a_{ij}(t)$ the number of packets arriving at input i at time slot t, and destined to output port j. We assume $a_{ij}(t)$ is a Bernoulli random variable with parameter λ_{ij}, and $a_{ij}(t)'$s are independent across time and input-port and output-port pairs. We further assume that one packet can be transferred from an input port to an output port in one time slot.

Define the arrival rate matrix λ to be an $N \times N$ matrix such that the $(i,j)^{\text{th}}$ entry is λ_{ij}. We say arrival rate matrix λ is *supportable* if there exists a switch scheduling under which

$$\lim_{t\to\infty} \Pr\left(|q(t|) < \infty\right) = 1. \tag{3.1}$$

So λ is supportable if the VOQs remain finite. We will first characterize the set of supportable arrival rate matrices.

Since at most one packet can be transferred from an input port in one time slot, a necessary condition for λ to be supportable is

$$\sum_{j=1}^{N} \lambda_{ij} \le 1,$$

i.e., the aggregate arrival rate to input port i should be no more than one (packet per time slot). Similarly, each output port can accept at most one packet in each time slot, so

$$\sum_{i=1}^{N} \lambda_{ij} \le 1,$$

i.e., the number packets destined to output port j cannot exceed one (packet per time slot) on average. Based on the two conditions above, we define set $\mathcal{C}$ to be

$$\mathcal{C} = \left\{ \lambda : \lambda \geq 0, \sum_{i=1}^{N} \lambda_{ij} \leq 1 \text{ for any } j \text{ and } \sum_{j=1}^{N} \lambda_{ij} \leq 1 \text{ for any } i \right\}.$$

We will see in this section that $\mathcal{C}$ is the capacity region of the switch, and any λ that lies strictly inside $\mathcal{C}$ can be supported by a scheduling algorithm, called the MaxWeight scheduling algorithm.

The first result we will present is that no scheduling algorithm can support λ if λ is not in $\mathcal{C}$.

Theorem 3.2.1 *If $\lambda \notin \mathcal{C}$, then no scheduling algorithm can support arrival rate matrix λ.*

Proof Let $M(t)$ denote the schedule used in time slot t. So $M(t)$ is a $N \times N$ matrix such that $M_{ij}(t) = 1$ if input port i is connected to output port j in time slot t, and $M_{ij}(t) = 0$ otherwise.

According to the definition of $\mathcal{C}$, $\lambda \notin \mathcal{C}$ implies that either $\sum_i \lambda_{ij} > 1$ for some j or $\sum_j \lambda_{ij} > 1$ for some i. Suppose the first case occurs and there exist j^* and $\epsilon > 0$ such that $\sum_i \lambda_{ij^*} \geq 1 + \epsilon$. In this case, we consider the value of $\sum_i q_{ij^*}(t+1)$. Note that

$$q_{ij}(t+1) = (q_{ij}(t) + a_{ij}(t) - M_{ij}(t))^+ \geq q_{ij}(t) + a_{ij}(t) - M_{ij}(t),$$

so

$$\sum_i q_{ij^*}(t+1) \geq \sum_{s=1}^{t} \left(\sum_i a_{ij^*}(s) - \sum_i M_{ij^*}(s) \right).$$

According to the Strong Law of Large Numbers (SLLN), with probability one,

$$\lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} \sum_i a_{ij^*}(s) = \sum_i \lambda_{ij^*} \geq 1 + \epsilon.$$

Further, since $M(t)$ is a matching, $\sum_i M_{ij^*}(t) \leq 1$ for all t. So we have

$$\frac{1}{t} \sum_{s=1}^{t} \sum_i M_{ij^*}(s) \leq 1.$$

Therefore, with probability one,

$$\sum_i q_{ij^*}(t) \to \infty \quad \text{as} \quad t \to \infty.$$

So λ is not supportable. The same proof works if $\sum_j \lambda_{ij} \geq 1$ for some i.

□

The theorem above shows that any arrival rate matrix outside of $\mathcal{C}$ cannot be supported. We next present an algorithm that supports an arrival rate matrix that lies strictly in $\mathcal{C}$. Recall that an $N \times N$ switch can be represented by an $N \times N$ complete bipartite graph. Let H denote the total number of matchings in an $N \times N$ complete bipartite graph, and $M^{(h)}$ denote the h^{th} matching,

where we use $(\cdot)$ to indicate that the superscript is an index not power. Note that $M^{(h)}$ is an $N \times N$ matrix such that $M_{ij}^{(h)} = 1$ if input port i is connected to output port j and $M_{ij}^{(h)} = 0$ otherwise. We now present the MaxWeight scheduling algorithm, which will be shown to support any λ strictly within the capacity region $\mathcal{C}$. We now introduce the MaxWeight scheduling algorithm that can support any λ such that $(1+\epsilon)\lambda \in \mathcal{C}$. The MaxWeight scheduling algorithm associates a weight $q_{ij}^{(h)}$ with the link corresponding to VOQ(i,j) in the bipartite graph representation of the switch. It then finds a matching which maximizes the sum of the weights of the links included in the matching; hence the name MaxWeight algorithm.

MaxWeight Scheduling: The switch finds a matching $M(t)$ such that

$$M(t) \in \arg\max_{M^{(h)}} \sum_{ij} q_{ij}(t) M_{ij}^{(h)},$$

and transfers a packet from VOQ(i,j) to output port j if $M_{ij}(t) = 1$ and $q_{ij}(t) + a_{ij}(t) > 0$, i.e., there is a packet in the queue.

Queue length $q_{ij}(t)$ can be thought as the weight associated with edge (i,j) in the complete bipartite graph representing the switch. Therefore, the MaxWeight algorithm finds a matching with the maximum sum weight among all matchings, so is called *MaxWeight scheduling.*

Example 8 *Consider a* 3×3 *cross-bar switch, with the states of VOQs are shown in Figure 3.7. MaxWeight scheduling will schedule the two links shown on the left. This schedule has a sum weight of* 4*. The matching on the right has a sum weight equal to* 3*, so will not be selected by the algorithm.* □

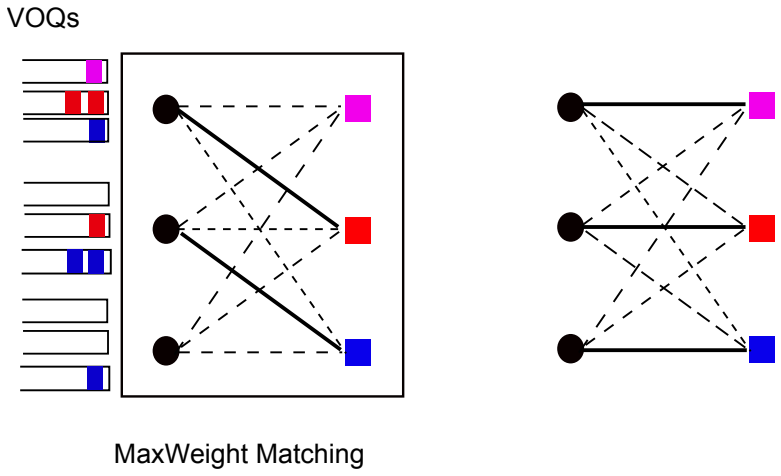

Figure 3.7: An example of MaxWeight scheduling in a 3×3 switch

To analyze the performance of MaxWeight scheduling, we need the Birkhoff-von Neumann theorem which establishes the connection between doubly stochastic matrices and permutation matrices.

Definition 3.2.1 (Doubly Stochastic Matrix) *An $N \times N$ matrix λ is a doubly stochastic matrix if $0 \le \lambda_{ij} \le 1$, $\sum_{i=1}^{N} \lambda_{ij} = 1$, and $\sum_{j=1}^{N} \lambda_{ij} = 1$.* □

Definition 3.2.2 (Permutation Matrix) *An $N \times N$ matrix U is a permutation matrix if $\sum_{i=1}^{N} U_{ij} = 1$, $\sum_{j=1}^{N} U_{ij} = 1$ and $U_{ij} \in \{0, 1\}$.* □

Lemma 3.2.2 (Birkhoff-von Neumann Theorem) *An $N \times N$ matrix λ is a doubly stochastic matrix if and only if it is a convex combination of permutation matrices, i.e., there exists $\beta \ge 0$ such that $\sum_h \beta_h = 1$ and $\lambda = \sum_h \beta_h U^{(h)}$, where $U^{(h)}$s are permutation matrices.* □

We comment that a doubly stochastic matrix λ belongs to set $\mathcal{C}$, and a permutation matrix is a valid matching.

Theorem 3.2.3 *MaxWeight scheduling can support any arrival rate matrix λ such that $(1+\epsilon)\lambda \in \mathcal{C}$.*

Proof Note that $q(t)$ is an irreducible Markov chain under MaxWeight scheduling (see Exercise 2 for the details). We prove this theorem by demonstrating that $q(t)$ is positive recurrent. Defining the Lyapunov function to be

$$V(q(t)) = \sum_{ij} q_{ij}^2(t),$$

we next will prove that if $(1+\epsilon)\lambda \in \mathcal{C}$ for some $\epsilon > 0$, then

$$E\left[V(q(t+1)) - V(q(t))\right| q(t) = q] \le \left(\sum_{ij} \lambda_{ij}\right) + N - 2\epsilon \sum_{i,j} \lambda_{ij} q_{ij}. \tag{3.2}$$

Therefore, the theorem follows from the Foster-Lyapunov theorem (Theorem 2.3.7).

To prove result (3.2), we first have

$$\begin{aligned}
& V(q(t+1)) - V(q(t)) \\
= & \sum_{ij} \left((q_{ij}(t) + a_{ij}(t) - M_{ij}(t))^+\right)^2 - \sum_{ij} q_{ij}^2(t) \\
\le & \sum_{ij} \left(q_{ij}(t) + a_{ij}(t) - M_{ij}(t)\right)^2 - \sum_{ij} q_{ij}(t) \\
= & \sum_{ij} \left(2q_{ij}(t) + a_{ij}(t) - M_{ij}(t)\right)\left(a_{ij}(t) - M_{ij}(t)\right) \\
= & \sum_{ij} 2q_{ij}(t)\left(a_{ij}(t) - M_{ij}(t)\right) + \sum_{ij} \left(a_{ij}(t) - M_{ij}(t)\right)^2.
\end{aligned}$$

Note that

$$\begin{aligned}
E\left[\sum_{ij} \left(a_{ij}(t) - M_{ij}(t)\right)^2 \middle| q(t) = q\right] & \le E\left[\sum_{ij} \left(a_{ij}^2(t) + M_{ij}^2(t)\right) \middle| q(t) = q\right] \\
& = \left(\sum_{ij} \lambda_{ij}\right) + N,
\end{aligned}$$

where $E[a_{ij}^2(t)|q(t)=q]=\lambda_{ij}$ because $a_{ij}(t)$ is Bernoulli and independent of $q(t)$, and $E[\sum_{ij} M_{ij}^2(t)|q(t)=q]\leq N$ because $M(t)$ is a matching. So taking conditional expectations on both sides of the equation above, we obtain

$$E\left[V(q(t+1))-V(q(t))|\,q(t)=q\right]\leq\left(\sum_{ij}\lambda_{ij}\right)+N+2\sum_{ij}q_{ij}\left(\lambda_{ij}-E[M_{ij}(t)|q(t)=q]\right). \quad (3.3)$$

Since there exists $\epsilon>0$ such that $(1+\epsilon)\lambda\in\mathcal{C}$. According to the Birkhoff-von Neumann theorem, there exists a vector $\beta\geq 0$ such that $\sum_h\beta_h=1$ and $(1+\epsilon)\lambda\leq\sum_h\beta_h U^{(h)}$, where $U^{(h)}$s are permutation matrices and $\leq$ means element-wise inequalities here. Therefore,

$$\begin{aligned}(1+\epsilon)\sum_{ij}q_{ij}\lambda_{ij} &\leq \sum_{ij}q_{ij}\left(\sum_h\beta_h U_{ij}^{(h)}\right)=\sum_h\beta_h\left(\sum_{ij}q_{ij}U_{ij}^{(h)}\right)\\ &\leq \left(\sum_h\beta_h\right)\left(\max_h\left(\sum_{ij}q_{ij}U_{ij}^{(h)}\right)\right)\\ &= \max_h\left(\sum_{ij}q_{ij}U_{ij}^{(h)}\right).\end{aligned}$$

Since permutation matrices are valid matchings, we further have

$$(1+\epsilon)\sum_{ij}q_{ij}\lambda_{ij}\leq\max_h\sum_{ij}q_{ij}U_{ij}^{(h)}\leq\max_h\sum_{i,j}q_{ij}M_{ij}^{(h)}=\sum_{i,j}q_{ij}E[M_{ij}(t)|q(t)=q],$$

where the last equality yields from the definition of MaxWeight scheduling. Substituting the inequality above into inequality (3.3) yields

$$E\left[V(q(t+1))-V(q(t))|\,q(t)=q\right]\leq\left(\sum_{ij}\lambda_{ij}\right)+N-2\epsilon\sum_{i,j}\lambda_{ij}q_{ij}. \quad (3.4)$$

Hence, inequality (3.2) holds, and the proof completes.

□

From Theorems 3.2.1 and 3.2.3, we conclude that any arrival rate matrix λ not in $\mathcal{C}$ cannot be supported by any switch scheduling algorithm, while any λ that is *strictly* in $\mathcal{C}$ can be supported by MaxWeight scheduling. Therefore, $\mathcal{C}$ is called the *capacity region* of an $N\times N$ switch, and MaxWeight scheduling is said to be a *throughput optimal* algorithm.

Chapter 4

Scheduling in Wireless Networks

So far we have looked at resource allocation algorithms in networks with wireline links. In this chapter, we consider networks with wireless components. The major difference between wireless and wireline networks is that, in wireless networks, links contend for a common resource, namely, the wireless spectrum. As a result, we have to design Medium Access Control (MAC) algorithms to decide which links access the wireless medium at each time instant. As we will see, wireless MAC algorithms share features similar to scheduling algorithms in the switch fabric of a high-speed router, which were studied in the previous chapter. However, there are some differences: wireless networks are subject to time-varying link quality due to channel fluctuations, also known as channel fading. In addition, some wireless networks do not have a central coordinator to perform scheduling, so scheduling decisions have to be taken independently by each link. We will address these issues specific to wireless networks in this chapter.

4.1 Channel-Aware Scheduling in Cellular Networks

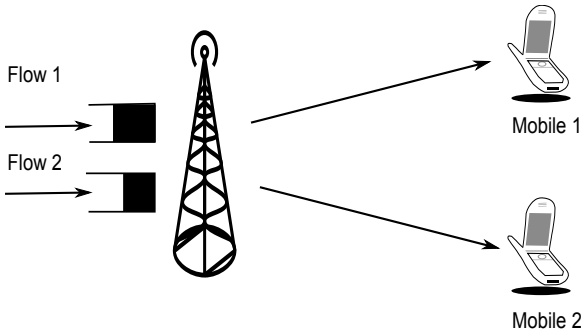

Figure 4.1: A downlink network with a single base station and two mobiles

We first use a simple example to demonstrate the importance of considering channel state information in scheduling. Consider a downlink network with a single base-station and two mobiles as shown in Figure 4.1. Assume the system is time-slotted. The base station has only one antenna, and can only transmit to one mobile at a time. So we do not deal with interference in this example.

We further assume the two channels are independent ON-OFF channels, and each channel is ON with probability 1/2. When a channel is ON, one packet can be transmitted over the channel. This ON-OFF channel model is a abstraction of a situation where the quality of the channels are time-varying due to fading. Suppose we assume that the base station uses a fix transmit power, and that mobiles can successfully decode a packet transmission only when the Signal to Noise Ratio (SNR) is above a threshold. (Note that SNR is the same as SINR discussed earlier when the interference power is zero.) Then, the ON state corresponds to the state of the channel when the SNR is above the desired threshold. Since $h_i(t)$ is time-varying, the channel can be modeled as an ON-OFF channel.

Next we consider two scheduling policies, a channel-unaware policy and a channel-aware scheduling policy, to demonstrate the importance of including channel state information in scheduling. The channel unaware scheduling is presented below, which schedules the mobiles independent of their channel states.

Channel Unaware Scheduling: The base station selects mobile i with probability α_i, and transmits a packet to mobile i.

Under this channel unaware scheduling, the rate allocated to mobile i is

$$\alpha_i \Pr(\text{channel } i \text{ is on}) = \frac{\alpha_i}{2} \quad \text{(packets/slot)}.$$

Thus, by varying α_1 and α_2, any service rate vector of the form $(\alpha_1/2, \alpha_2/2)$ can be achieved by the channel unaware scheduling algorithm, where $\alpha_1 + \alpha_2 \leq 1$.

The channel aware scheduling is presented below, where the base station schedules the mobiles based on their channel states.

Channel Aware Scheduling: Assume that the base station knows the states of both channels through the feedback from the mobiles. If only one mobile's channel is ON, the base station schedules the mobile with ON channel; otherwise, the base station schedules mobile i with probability α_i.

Under this scheduling algorithm, mobile 1 (2) is scheduled with probability one when its channel is ON, and mobile 2's (1's) channel is OFF; and mobile 1 is scheduled with probability α_i when both channels are ON. Therefore, the rate allocated to mobile i ($i = 1, 2$) is

$$\begin{aligned} & \Pr(\text{channel } i \text{ is ON, the other channel if OFF}) + \alpha_i \Pr(\text{both channels are ON}) \\ = \; & \frac{1}{4} + \alpha_i \frac{1}{4} \\ = \; & \frac{(1+\alpha_i)}{4}. \end{aligned}$$

The set of rate allocations (μ_1, μ_2) such that

$$\mu_1 \leq \frac{(1+\alpha_1)}{4} \text{ and } \mu_2 \leq \frac{(1+\alpha_2)}{4} \tag{4.1}$$

is feasible under the channel aware scheduling.

By varying α_1 and α_2, we can plot the capacity regions of both scheduling algorithms, as shown in Figure 4.2. The capacity region of channel aware scheduling is clearly larger than that

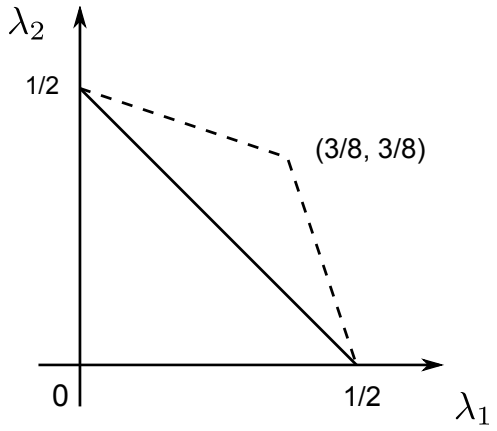

Figure 4.2: The region $Co\{(0,0),(0,1/2),(1/2,0)\}$ is the capacity region of the channel unaware scheduling, and the region $Co\{(0,0),(0,1/2),(3/8,3/8),(1/2,0)\}$ is the capacity region of the channel aware scheduling.

of channel unaware scheduling. This example demonstrates the importance of exploiting channel state information in scheduling. To implement channel-aware scheduling, one needs to know the state of the channels. We will assume that such channel-state information can be obtained without much overhead.

4.2 The MaxWeight Algorithm for the Cellular Downlink

Motivated by the simple example above, in this section, we consider channel aware scheduling in a general cellular downlink network with a single base station and N mobiles. We further make the following assumptions:

(i) Assume the network has M channel states, indexed by m, where the channel state refers to the states of all channels in the network.

(ii) The base station can transmit to one mobile at a time.

(iii) Define c_m to be the channel rate vector when the channel is in state m, where $c_{m,i}$ is the transmission rate to mobile i, if it is scheduled. Assume there exists $c_{\max} < \infty$ such that $c_{m,i} \leq c_{\max}$ for all i and m. Let $\mu_i(t)$ denote the transmission rate to mobile i at time t, so $\mu_i(t) = c_{m,i}$ if mobile i is scheduled in state m.

(iv) The channel state process is i.i.d. across time slots. The probability that the channel is in state m is denoted by π_m.

(v) Packet arrival processes are independent across users and time slots. Let $a_i(t)$ denote the the number of packet arrivals for mobile i at time slot t, which is a random variable that takes positive integer values, with mean λ_i and a finite variance σ_i^2 for user i. We further assume $\Pr(a_i(t) = 0) > 0$ for all i.

(vi) The base station maintains a separate queue for each mobile. Let $q_i(t)$ denote the number of packets buffered at the base station for mobile i at the beginning of time slot t, before arrivals occur.

The assumptions that the base station can transmit to one mobile at a time, i.i.d. channel fading, and i.i.d. arrival processes can be relaxed. We impose these assumptions to simplify the analysis.

Similar to switch scheduling, arrival rate vector λ is said to be *supportable* if there exists a scheduling algorithm under which

$$\lim_{t\to\infty} \Pr\left(q(t) < \infty\right) = 1.$$

Next we first characterize the capacity region of the cellular downlink defined above, and then present a scheduling algorithm that supports any arrival rate vector that lies strictly in the capacity region.

Definition 4.2.1 (Capacity Region $\mathcal{C}$) *An arrival rate vector λ is in $\mathcal{C}$ if there exists $\{\alpha_{m,i} \geq 0\}$ (interpret $\alpha_{m,i}$ as the probability user i is scheduled in channel state m) such that*

$$\lambda_i \leq \sum_{m=1}^{M} \alpha_{m,i} c_{m,i} \pi_m, \quad \forall i \tag{4.2}$$

and

$$\sum_i \alpha_{m,i} \leq 1 \quad \forall m.$$

□

Note the set $\mathcal{C}$ satisfies the property that given any $\lambda \in \mathcal{C}$, $\lambda^{(i)}$, such that $\lambda_i^{(i)} = 0$ and $\lambda_j^{(i)} = \lambda_j$ for $j \neq i$, also belongs to $\mathcal{C}$. A convex set with this property is called *coordinate convex.*

Example 9 *Consider the network with single base station and two mobiles, presented in Section 4.1. The network has four states:* $(0,0)$ *(both channels are OFF),* $(0,1)$ *(channel* 1 *is off and channel* 2 *is ON),* $(1,0)$ *(channel* 1 *is on and channel* 2 *is OFF), and* $(1,1)$ *(both channels are ON). Since a channel is equally likely to be ON or OFF, we have*

$$\pi_{(0,0)} = \pi_{(0,1)} = \pi_{(1,0)} = \pi_{(1,1)} = 0.25.$$

The capacity region $\mathcal{C}$ is the set of λ's such that

$$\lambda_1 \leq \alpha_{(0,0),1} \times 0 \times \pi_{(0,0)} + \alpha_{(0,1),1} \times 0 \times \pi_{(0,1)} + \alpha_{(1,0),1} \times 1 \times \pi_{(1,0)} + \alpha_{(1,1),1} \times 1 \times \pi_{(1,1)}$$
$$\lambda_2 \leq \alpha_{(0,0),2} \times 0 \times \pi_{(0,0)} + \alpha_{(0,1),2} \times 1 \times \pi_{(0,1)} + \alpha_{(1,0),2} \times 0 \times \pi_{(1,0)} + \alpha_{(1,1),2} \times 1 \times \pi_{(1,1)}.$$

Clearly, to maximize throughput, we should not schedule an OFF channel, so

$$\alpha_{(0,0),1} = \alpha_{(0,0),2} = \alpha_{(0,1),1} = \alpha_{(1,0),2} = 0$$
$$\alpha_{(0,1),2} = \alpha_{(1,0),1} = 1.$$

Therefore, the capacity region can be written as

$$\lambda_1 \leq \pi_{(1,0)} + \alpha_{(1,1),1}\pi_{(1,1)} = \frac{1+\alpha_{(1,1),1}}{4}$$
$$\lambda_2 \leq \pi_{(0,1)} + \alpha_{(1,1),2}\pi_{(1,1)} = \frac{1+\alpha_{(1,1),2}}{4},$$

which is equivalent to (4.1). □

The following theorem shows that any $\lambda \notin \mathcal{C}$ is not supportable.

Theorem 4.2.1 *No scheduling algorithm can support arrival rate vector $\lambda \notin \mathcal{C}$. .*

Proof Note that $\mathcal{C}$ is convex. According to the Strict Separation Theorem (Theorem 1.1.4), if $\lambda \notin \mathcal{C}$, then there exist $\beta \in \mathcal{R}^N$, $\beta \neq 0$ and $\delta > 0$ such that

$$\sum_i \beta_i \lambda_i \geq \sum_i \beta_i x_i + \delta \tag{4.3}$$

for any $x \in \mathcal{C}$.

Now suppose $\beta \geq 0$. Define $V(t) = \sum_i \beta_i q_i(t)$. Note that

$$q_i(t+1) = (q_i(t) + a_i(t) - \mu_i(t))^+ \geq q_i(t) + a_i(t) - \mu_i(t),$$

which implies that

$$V(t+1) \quad \geq \quad \sum_{s=1}^{t} \sum_i \beta_i \left(a_i(s) - \mu_i(s)\right).$$

Denote by $m(t)$ the channel state at time t. According to the SLLN, with probability one,

$$\lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} \sum_i \beta_i a_i(s) \quad = \quad \sum_i \beta_i \lambda_i$$
$$\lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} \mathbb{I}_{m(s)=l} \quad = \quad \pi_l.$$

Due to inequality (4.3), we can further obtain that with probability one,

$$\lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} \sum_i \beta_i a_i(s) \geq \lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} \beta_i \mu_i(s) + \delta,$$

which implies that $\lim_{t\to} V(t) = \infty$ with probability one. So λ is not supportable.

We now prove that there exists a $\beta \geq 0$ satisfying condition (4.3) to complete the theorem. Given any β that satisfies (4.3), we define $\tilde{\beta}$ to be $\tilde{\beta}_i = \beta_i$ if $\beta_i \geq 0$ and $\tilde{\beta}_i = 0$ otherwise. Since $\lambda \geq 0$, we first have

$$\sum_i \tilde{\beta}_i \lambda_i \geq \sum_i \beta_i \lambda_i. \tag{4.4}$$

Next, given any $x \in \mathcal{C}$, we define $\tilde{x}$ such that $\tilde{x}_i = x_i$ if $\beta_i \geq 0$ and $\tilde{x}_i = 0$ otherwise. So we have $\tilde{\beta}_i x_i = \beta_i \tilde{x}_i$, and

$$\sum_i \tilde{\beta}_i x_i = \sum_i \beta_i \tilde{x}_i \leq \sum_i \beta_i \lambda_i, \tag{4.5}$$

where the last inequality holds because according to the definition of $\mathcal{C}$, $\tilde{x}$ also belongs to $\mathcal{C}$ given $x \in \mathcal{C}$. Combining inequalities (4.4) and (4.5), we conclude that $\tilde{\beta}$ is a nonnegative vector that satisfies condition (4.3).

□

If the average arrival rate λ and channel state distribution π are known, we can choose α that satisfies (4.2), and schedule the mobiles according to the channel state and α. Knowing arrival rate λ and channel state distribution π, however, is difficult in reality. The question, therefore, is *can we design a traffic-blind policy, such as MaxWeight for high-speed switches, for wireless networks?*

We note that scheduling in wireless networks has some similarities with that in high-speed switches. In fact, if the channel state is fixed instead of time varying, then a downlink network can be viewed as a switch with single input port and N output ports. So a natural question is whether a variation of the MaxWeight algorithm can be used to achieve the maximum throughput in cellular networks. We next present the MaxWeight algorithm for wireless downlink networks, and prove its throughput optimality.

MaxWeight Scheduling for Downlink Networks: At each time instant t, the base station transmits to mobile i such that

$$i \in \arg\max_j c_{m(t),j} q_j(t)$$

with rate $c_{m(t),i}$, breaking ties at random.

Example 10 *Again, consider the network with single base station and two mobiles, presented in Section 4.1, which has four states:*

- *When both channels are off, no mobile is scheduled.*
- *When channel* 1 *is off and channel* 2 *is on,* $c_1(t)q_1(t) = 0 \leq c_2(t)q_2(t) = q_2(t)$*, so mobile* 2 *is scheduled.*
- *When channel* 1 *is on and channel* 2 *is off,* $c_1(t)q_1(t) = q_1(t) \geq c_2(t)q_2(t) = 0$*, so mobile* 1 *is scheduled.*
- *When both channels are on,* $c_1(t)q_1(t) = q_1(t)$ *and* $c_2(t)q_2(t) = q_2(t)$*, so the base station schedules the mobile with longer queue length.*

□

In the next theorem, we show that MaxWeight scheduling can stabilize any λ that lies strictly inside set $\mathcal{C}$.

Theorem 4.2.2 *Given any arrival rate vector* λ *such that* $(1+\epsilon)\lambda \in \mathcal{C}$ *for some* $\epsilon > 0$*,* $q(t)$ *is a positive recurrent under the MaxWeight algorithm.*

Proof It is easy to verify $q(t)$ is an irreducible Markov chain under the assumption that $\Pr(a_i(t) = 0) > 0$ all i. We first prove the following fact, which is the key to this proof. Given $q(t) = q$, under MaxWeight scheduling,

$$E\left[\sum_i q_i \mu_i(t)\right] \geq \sum_i q_i \gamma_i \quad \forall\ \gamma \in \mathcal{C}. \tag{4.6}$$

According to definition of $\mathcal{C}$, for any $\gamma \in \mathcal{C}$, there exists α such that $\gamma_i \leq \sum_{m=1}^{M} \alpha_{m,i} c_{m,i} \pi_m$ for all i. So

$$\begin{aligned}
\sum_i q_i \gamma_i &\leq \sum_i q_i \left(\sum_{m=1}^{M} \alpha_{m,i} c_{m,i} \pi_m\right) \\
&= \sum_{m=1}^{M} \pi_m \left(\sum_i q_i \alpha_{m,i} c_{m,i}\right) \\
&\leq \sum_{m=1}^{M} \pi_m \left(\max_i q_i c_{m,i}\right),
\end{aligned}$$

where the last inequality holds because $\sum_i \alpha_{m,i} \leq 1$ for all i. Now given $q(t) = q$ and $c(t) = c_m$, MaxWeight scheduling selects a mobile with the largest $q_i c_{m,i}$, so

$$E\left[\sum_i q_i \mu_i(t)\right] = \sum_{m=1}^{M} \pi_m \left(\max_i q_i c_{m,i}\right) \geq \sum_i q_i \gamma_i,$$

and (4.6) holds.

Define the Lyapunov function to be

$$V(q(t)) = \sum_{i=1}^{N} q_i^2(t).$$

The conditional drift of the Lyapunov function is

$$\begin{aligned}
& E\left[V(q(t+1)) - V(q(t)) | q(t) = q\right] \\
= \ & E\left[\sum_{i=1}^{N} (q_i^2(t+1) - q_i^2(t)) \middle| q(t) = q\right] \\
= \ & E\left[\sum_{i=1}^{N} \left((q_i(t) + a_i(t) - \mu_i(t))^+\right)^2 - q_i^2(t) \middle| q(t) = q\right] \\
\leq \ & E\left[\sum_{i=1}^{N} (q_i(t) + a_i(t) - \mu_i(t))^2 - q_i^2(t) \middle| q(t) = q\right] \\
= \ & E\left[\sum_{i=1}^{N} (a_i(t) - \mu_i(t))^2 + 2q_i(t)(a_i(t) - \mu_i(t)) \middle| q(t) = q\right] \qquad (4.7) \\
= \ & E\left[\sum_{i=1}^{N} (a_i(t) - \mu_i(t))^2 \middle| q(t) = q\right] + 2\sum_i q_i \left(\lambda_i - E\left[\mu_i(t) | q(t) = q\right]\right) .. \qquad (4.8)
\end{aligned}$$

Since the arrival processes are assumed to be independent of the queue state,

$$\begin{aligned} & E\left[\sum_{i=1}^{N}(a_i(t)-\mu_i(t))^2\middle| q(t)=q\right] \\ = \; & E\left[\sum_{i=1}^{N}a_i^2(t)-2a_i(t)\mu_i(t)+\mu_i^2(t)\middle| q(t)=q\right] \\ \leq \; & \sum_{i=1}^{N}\left(\sigma_i^2+c_{\max}^2\right). \end{aligned} \tag{4.9}$$

Since there exists $\epsilon > 0$ such that $(1+\epsilon)\lambda \in \mathcal{C}$, based on the fact (4.6), we have

$$\sum_i q_i(1+\epsilon)\lambda_i \leq \sum_i q_i E\left[\mu_i(t)|q(t)=q\right]. \tag{4.10}$$

Substituting inequalities (4.9) and (4.10) into (4.8), we obtain

$$E\left[V(q(t+1))-V(q(t))|q(t)=q\right] \leq \sum_{i=1}^{N}\left(\sigma_i^2+c_{\max}^2\right)-2\epsilon\sum_{i=1}^{N}q_i\lambda_i.$$

So the Markov chain $q(t)$ is positive recurrent according to Theorem 2.3.8, and the theorem holds.

□

4.3 MaxWeight Scheduling Ad Hoc P2P Wireless Networks

Ad hoc wireless networks refer to wireless networks in which wireless devices directly communicate with each other, instead of going through base stations or access points. While wireless access networks, including cellular networks and Wireless Local Area Networks (WLAN), are most common today; ad hoc wireless networks have emerged for a wide range of applications such as city-wide wireless mesh networks, wireless sensor networks, and vehicular networks.

In this and the next section, we will focus on modeling and design of scheduling algorithms for ad hoc wireless networks. We will assume that the ad hoc network operates in a peer-to-peer fashion, i.e., a source node directly transmits a packets to its destination without using multi-hop routing. Multi-hop wireless networks will be considered in the next chapter.

Figure 4.3 shows an example of an ad hoc P2P network, where nodes are distributed in a two-dimensional space, and form multiple source-destination pairs (called links). We will focus our attention on resolving interference in ad hoc wireless networks by assuming all channels are time-invariant channels, i.e., no fading. We further model the interference as a conflict graph in which each node represents a wireless link and an edge from node a to node b means link a interferes link b, so link a should keep silent when link b is active. For example, if only non-adjacent links can transmit simultaneously, then the conflict graph associated with the network in Figure 4.3 is shown in Figure 4.4.

Note that a collection of nodes (which are wireless links in the real network) in the conflict graph can be scheduled simultaneously if they do not interfere with each other. Such a set of nodes is called a *schedule* and because of the importance of this concept, we present it as a definition below.

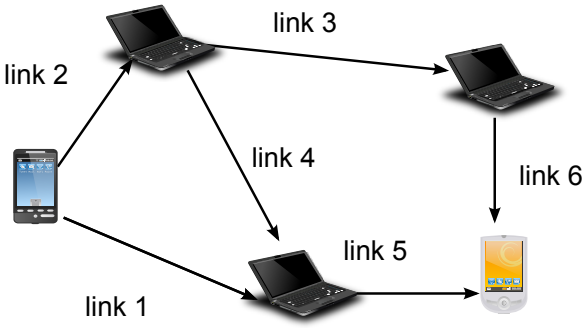

Figure 4.3: An ad hoc P2P wireless network

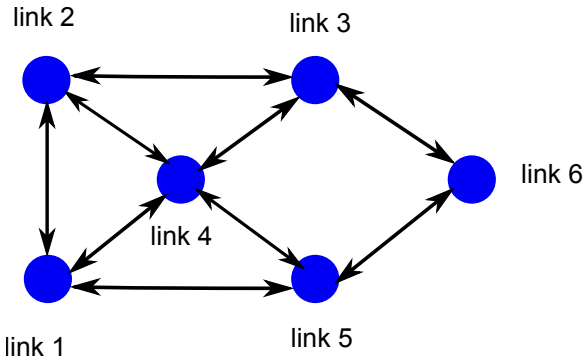

Figure 4.4: A conflict graph associated with the network in Figure 4.3

Definition 4.3.1 (Schedule) *A set of links that can be active simultaneously without interfering each other.* □

Consider the conflict graph in Figure 4.4, the link set $\{1, 3\}$ is an schedule. The scheduling problem can be thought of as finding a good schedule in the conflict graph at each time instant.

Inspired by the MaxWeight algorithm for high-speed switches and for the cellular downlink, we are interested in the performance of MaxWeight scheduling in ad hoc P2P wireless networks. For simplicity, we assume Bernoulli arrivals, and one packet can be transmitted in one time slot if a link is on. Let $q_l(t)$ denote the number of packets queued at the transmitter of link l. We also assume the conflict graph has H distinct schedules. Let $M^{(1)}, M^{(2)}, \ldots, M^{(H)}$ represent the H schedules, where each M is a vector of size L and L is the number of links in the network. According to this definition,

$$M_l^{(h)} = \begin{cases} 1 & \text{if link } l \text{ is in schedule } M^{(h)} \\ 0 & \text{else.} \end{cases}$$

The MaxWeight algorithm for ad hoc P2P wireless networks is presented below, where in each time instant, a schedule with the largest sum queue length is used, i.e., the links in the schedule transmit and all other links keep silent.

MaxWeight Scheduling for Ad Hoc P2P Wireless Networks: At each time instant t, the network schedules the links in schedule $M(t)$ such that

$$M(t) \in \arg\max_{M^{(h)}} \sum_l M_l^{(h)} q_l(t). \tag{4.11}$$

Break ties arbitrarily.

Example 11 *Consider a conflict graph shown in Figure 4.5. MaxWeight scheduling will schedule* $\{4, 6\}$, *which has the maximum sum weight* 21. □

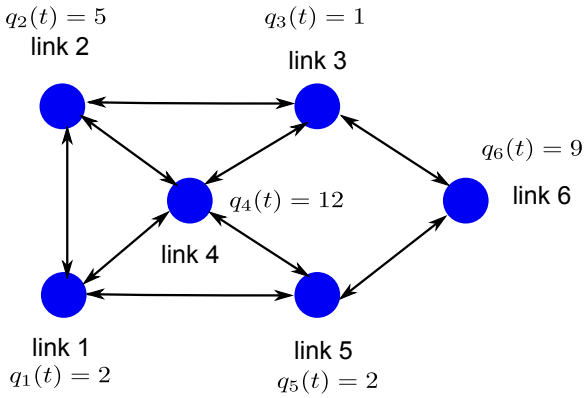

Figure 4.5: A conflict graph

We next define the set $\mathcal{C}$ to be

$$\mathcal{C} = \left\{ \lambda : \lambda_l \leq \sum_{h=1}^{H} \alpha_h M_l^{(h)} \ \forall l, \quad \text{for some } \alpha \geq 0 \text{ such that } \sum_h \alpha_h \leq 1 \right\},$$

where α_h can be viewed as the probability to schedule $M^{(h)}$. The next theorem says that $\mathcal{C}$ is the capacity region of the ad hoc P2P wireless network.

Theorem 4.3.1 *No scheduling algorithm can support arrival rate vector* $\lambda \notin \mathcal{C}$.

Proof The proof is based on the strict separation theorem and SLLN, and is similar to the proof of Theorem 4.2.1.

□

We now prove that the MaxWeight algorithm is throughput optimal in ad hoc P2P wireless networks.

Theorem 4.3.2 *Given an arrival rate vector* λ *such that* $(1 + \epsilon)\lambda \in \mathcal{C}$ *for some* $\epsilon > 0$, $q(t)$ *is positive recurrent under MaxWeight scheduling.*

Proof According to the definition of $\mathcal{C}$, given any $\gamma \in \mathcal{C}$, there exists $\alpha \geq 0$ such that $\sum_h \alpha \leq 1$ and

$$\gamma \leq \sum_h \alpha_h M^{(h)}.$$

So given $q(t) = q$,

$$\begin{aligned} \sum_l q_l \gamma_l &\leq \sum_l q_l \left(\sum_h \alpha_h M_l^{(h)} \right) = \left(\sum_h \alpha_h \right) \left(\sum_l q_l M_l^{(h)} \right) \\ &\leq \max_h \sum_l q_l M_l^{(h)} = \sum_l q_l M_l(t), \end{aligned}$$

where the last equality follows from the definition of MaxWeight scheduling.

If there exists $\epsilon > 0$ such that $(1+\epsilon)\lambda \in \mathcal{C}$, then given $q(t) = q$,

$$\sum_l q_l (1+\epsilon)\lambda_l \leq \sum_l q_l M_l(t),$$

and

$$\sum_l q_l \lambda_l - \sum_l q_l M_l(t) \leq -\epsilon \sum_l q_l \lambda_l.$$

Considering Lyapunov function $V(q(t)) = \sum_l q_l^2(t)$ and following the argument given in the proof of Theorem 4.2.2, we can prove that $q(t)$ is positive recurrent.

□

4.4 General MaxWeight Algorithms

We have now seen that the MaxWeight algorithm is a powerful scheduling mechanism, which is throughput optimal in the contexts of switch scheduling, cellular downlink networks and ad hoc P2P networks. The MaxWeight algorithm uses queue lengths as link weights so that if a flow does not receive enough service, its queue builds up, which forces the algorithm to allocate more resources to the flow. This interaction between queue lengths and scheduling guarantees the throughput optimality of resource allocation. The reader may wonder whether other choices of link weights can work as well. In this section, we will see that a large class of functions (of queue lengths) that are increasing and differentiable can be used as link weights while still maintaining throughput optimality.

Denoting by $w_l(q_l)$ the weight associated with link l, we present the general MaxWeight algorithm, in the setting of ad hoc peer-to-peer networks without fading. The algorithm can be easily extended to networks with fading channels.

General MaxWeight Scheduling for Ad Hoc P2P Wireless Networks: At time instant t, the network schedules the links in schedule $M(t)$ such that

$$M(t) \in \arg\max_{M^{(h)}} \sum_l w_l(q_l(t)) M_l^{(h)}.$$

If there is more than one such schedules, break ties arbitrarily.

We next introduce a class of weight functions, called valid weight functions, such that the general MaxWeight scheduling is throughput optimal if all link weights are valid weight functions.

Definition 4.4.1 (Valid Weight Functions for MaxWeight Scheduling) *We say a function $w(\cdot)$ is a* valid weight function *if the following conditions hold:*

(i) Function $w(x)$ is increasing and differentiable for $x \geq 0$, and $w(x) = 0$.

(ii) As $x \to \infty$, $w(x) \to \infty$.

(iii) Given any $\delta > 0$, there exist a constants $B_\delta \geq 0$ such that for any $x \geq 0$,

$$(1-\delta)w(x+1) - B_\delta \leq w(x) \leq (1+\delta)w\left((x-1)^+\right) + B_\delta. \tag{4.12}$$

□

Note that the third condition simply says that $w(x)$ does not change significantly when x increases or decreases by one. Specifically, when $w(x)$ is small, $|w(x) - w(x \pm 1)|$ is bounded by a constant, and when $w(x)$ is large, $\frac{w(x+1)}{w(x)}$ and $\frac{w(x)}{w(x-1)}$ are bounded by constants.

The following theorem states that the general MaxWeight scheduling is throughput optimal if all weight functions are valid weight functions. To simplify notation and analysis, we assume arrivals are Bernoulli.

Theorem 4.4.1 *Assume that the arrival processes to links in an ad hoc P2P network are Bernoulli processes. The general MaxWeight scheduling can support any λ such that $(1+\epsilon)\lambda \in \mathcal{C}$.*

Proof Following the proof of Theorem 4.3.1, given $q(t) = q$, the general MaxWeight algorithm guarantees that

$$(1+\epsilon)\sum_l w_l(q_l)\lambda_l \leq \sum_l w_l(q_l)M_l(t). \tag{4.13}$$

Now we need to choose a proper Lyapunov function to show that $q(t)$ is positive recurrent. We consider the following Lyapunov function:

$$V(q(t)) = \sum_l \int_0^{q_l(t)} w_l(x)\mathrm{d}x.$$

The conditional drift of the Lyapunov function is

$$\begin{aligned} & E\left[V(q(t+1)) - V(q(t))|q(t) = q\right] \\ = \; & E\left[\sum_l \left(\int_0^{q_l(t+1)} w_l(x)\,dx - \int_0^{q_l(t)} w_l(x)\,dx\right)\middle| q(t) = q\right] \\ = \; & E\left[\sum_l w_l(\tilde{q}_l)(q_l(t+1) - q_l(t))\middle| q(t) = q\right], \end{aligned}$$

where the last equality follows from the mean-value theorem and

$$\tilde{q}_l \in [\min(q_l(t), q_l(t+1)), \max(q_l(t), q_l(t+1))].$$

Note that if $q_l(t) + a_l(t) > 0$, then $q_l(t+1) = q_l(t) + a_l(t) - M_l(t)$; otherwise, $q_l(t+1) = q_l(t) = 0$ and $\tilde{q}_l = 0$. Therefore,

$$E\left[V(q(t+1)) - V(q(t))|q(t) = q\right] \leq E\left[\sum_l w_l(\tilde{q}_l)(a_l(t) - M_l(t))\middle| q(t) = q\right]. \tag{4.14}$$

Since $q_l(t)$ can at most increase or decrease by one in one time slot, $\tilde{q}_l \in [(q_l(t)-1)^+, q_l(t)+1]$. Since $w_l(x)$ is a valid weight function, it is increasing in x and satisfies condition (4.12). Therefore, given any $\delta > 0$, there exists $B_{l,\delta} > 0$ such that

$$w_l(\tilde{q}_l) \geq w_l((q_l(t)-1)^+) \geq \frac{w_l(q_l(t)) - B_{l,\delta}}{1+\delta}$$
$$w_l(\tilde{q}_l) \leq w_l(q_l(t)+1) \leq \frac{w_l(q_l(t)) + B_{l,\delta}}{1-\delta}.$$

Replacing $w_l(\tilde{q}_l)$ in inequality (4.14) using above two inequalities, we obtain

$$\begin{aligned}
& E\left[V(q(t+1)) - V(q(t))|q(t) = q\right] \\
\leq\ & \sum_l \frac{w_l(q_l) + B_{l,\delta}}{1-\delta}\lambda_l - \sum_l \frac{w_l(q_l) - B_{l,\delta}}{1+\delta} E[M_l(t)|q(t) = q] \\
\leq\ & K + \frac{1}{1-\delta}\sum_l w_l(q_l)\lambda_l - \frac{1}{1+\delta}\sum_l w_l(q_l) E[M_l(t)|q(t) = q] \\
=\ & K + \frac{1}{1+\delta}\left(\frac{1+\delta}{1-\delta}\sum_l w_l(q_l)\lambda_l - \sum_l w_l(q_l) E[M_l(t)|q(t) = q]\right),
\end{aligned}$$

where

$$K = \sum_l \frac{B_{l,\delta}\lambda_l}{1-\delta} + \sum_l \frac{B_{l,\delta}}{1+\delta}.$$

Now choose a small enough δ such that $(1+\delta)/(1-\delta) \leq 1 + \epsilon/2$. Then from inequality (4.13), we have

$$E\left[V(q(t+1)) - V(q(t))|q(t) = q\right] \leq K - \frac{\epsilon}{2}\frac{1}{1+\delta}\sum_l w_l(q_l)\lambda_l.$$

Since $w_l(q) \to \infty$ as $q \to \infty$, we can conclude that $q(t)$ is positive recurrent by invoking the Foster-Lyapunov theorem, which completes the proof.

□

So by selecting different weight functions, we can construct different throughput optimal scheduling algorithms. While all of these algorithms are throughput optimal, they may differ in their delay performances. However, the delay performance of these algorithms is hard to analyze and will not be considered here. We end this section by giving two examples of valid weight functions.

Example 12 *Weight function $w(q) = \kappa q^m$ for some $m > 0$ and $\kappa > 0$ is a valid weight function. Note that*

$$\kappa q^m = \frac{1}{(1+1/q)^m}\kappa(q+1)^m.$$

So given $\delta > 0$, there exists q_δ such that for any $q \geq q_\delta$,

$$\kappa q^m \geq (1-\delta)\kappa(q+1)^m.$$

So for any $q \geq 0$,

$$\kappa q^m \geq (1-\delta)\kappa(q+1)^m - (1-\delta)\kappa(q_\delta+1)^m.$$

Similarly, when $q \geq 1$,

$$\kappa q^m = \frac{1}{(1-1/q)^m}\kappa(q-1)^m.$$

So given $\delta > 0$, there exists $\tilde{q}_\delta$ such that for any $q \geq \tilde{q}_\delta$,

$$\kappa q^m \leq (1+\delta)\kappa(q-1)^m,$$

which implies that for any $q \geq 0$,

$$\kappa q^m \leq (1+\delta)\kappa((q-1)^+)^m + \kappa\tilde{q}_\delta^m.$$

Choosing $B_\delta = \max\{(1-\delta)\kappa(q_\delta+1)^m, \kappa\tilde{q}_\delta^m\}$, we conclude that κq^m is a valid weight function for $m > 0$ and $\kappa > 0$. □

Example 13 *Weight function $w(q) = \kappa\log(1+q)$ for some $\kappa > 0$ is a valid weight function. We note that*

$$\begin{aligned}\kappa\log(1+q+1) &= \kappa\log(1+q)\left(1+\frac{1}{1+q}\right)\\ &= \kappa\log(1+q)+\kappa\log\left(1+\frac{1}{1+q}\right)\\ &\leq \kappa\log(1+q)+\kappa\log 2.\end{aligned}$$

If $q \geq 1$, we have

$$\begin{aligned}\kappa\log(1+q-1) &= \kappa\log(1+q)+\kappa\log\left(1-\frac{1}{1+q}\right)\\ &\geq \kappa\log(1+q)-\kappa\log 2.\end{aligned}$$

If $0 \leq q < 1$, we have

$$\begin{aligned}\kappa\log\left(1+(q-1)^+\right) &= \kappa\log 1 = 0\\ &\geq \kappa\log(1+q)-\kappa\log 2.\end{aligned}$$

We therefore conclude that $\kappa\log(1+q)$ is a valid weight function by choosing $B_\delta = \kappa\log 2$. □

4.5 Q-CSMA: A Distributed Algorithm for Ad Hoc P2P Networks

The key step of the MaxWeight algorithm is to find the maximum weight independent set of the conflict graph. In wireless access networks, the base station (or access point) can collect channel and queue state information and solve the maximum weight independent set problem. In ad hoc P2P wireless networks, due to the lack of a centralized infrastructure, nodes have to make their own decisions on turning on or off based on local information, which requires us to seek distributed scheduling algorithms. In this section, we present the Queue-length-based CSMA/CA (Q-CSMA) algorithm for ad hoc P2P networks. Q-CSMA implements MaxWeight scheduling in a distributed fashion when link weights change slowly over time, so is throughput optimal.

4.5.1 The Idea behind Q-CSMA

Assume the conflict-graph of an ad hoc network has H independent sets. The key idea behind Q-CSMA is to select independent sets according to the following distribution:

$$\pi_h = \frac{1}{Z} \exp\left(\sum_l w_l M_l^{(h)}\right), \tag{4.15}$$

where w_l is the weight associated with link l and

$$Z = \sum_{h=1}^{H} \exp\left(\sum_l w_l M_l^{(h)}\right)$$

is the normalization factor. The reason to choose such a distribution is that the expected weight $E[\sum_l w_l M_l]$ under this distribution satisfies

$$E\left[\sum_l w_l M_l\right] \geq \max_h \sum_l w_l M_l^{(h)} - H, \tag{4.16}$$

which is close to the maximum weight when $\{w_l\}$ are sufficiently large. So when link weights are fixed and sufficiently large, if we generate a sufficient number of independent sets according to distribution (4.15), the algorithm performs like MaxWeight scheduling.

We next prove result (4.16). Define

$$\gamma_h = \sum_l w_l M_l^{(h)},$$

i.e., the sum weight associated with independent set $M^{(h)}$. Let $\mathcal{I}^*$ denote the set of maximum weight independent sets, i.e.,

$$\mathcal{I}^* = \left\{ h : \sum_l w_l M_l^{(h)} = \max_k \sum_l w_l M_l^{(k)} \right\},$$

and h^* denote an element of $\mathcal{I}^*$. The expected weight by selecting independent sets according to distribution (4.15) can be written as

$$\begin{aligned}
E\left[\sum_l w_l M_l\right] &= \sum_h \pi_h \gamma_h \\
&= \gamma_{h^*} - \sum_{h \notin \mathcal{I}^*} \frac{e^{\gamma_h}}{Z} (\gamma_{h^*} - \gamma_h) \\
&= \gamma_{h^*} - \frac{e^{\gamma_{h^*}}}{Z} \sum_{h \notin \mathcal{I}^*} \left(e^{\gamma_h - \gamma_{h^*}}\right) (\gamma_{h^*} - \gamma_h) \\
&\geq_{(a)} \gamma_{h^*} - \sum_{h \notin \mathcal{I}^*} 1 \\
&= \gamma_{h^*} - (H - |\mathcal{I}^*|) \\
&\geq \gamma_{h^*} - H,
\end{aligned}$$

where inequality (a) holds because $xe^{-x} \leq 1$ and $e^{\gamma_{h^*}}/Z \leq 1$.

4.5.2 Q-CSMA

We now present a distributed algorithm that generates independent sets according to distribution (4.15). We assume that w_l's are fixed and do not change with time. In reality, $w_l(q_l)$ will change, but if it changes very slowly, for example, if $w_l(q_l)$ is chosen to be slightly smaller than $\log(1+q_l)$, one can show that the stability results will not be affected, in manner that can be precisely described. We will not do so here. We will describe a DTMC whose states are the independent sets, and show that the steady-state distribution of this DTMC has the desired form. We will then describe a distributed algorithm under which the MAC layer behaves like the DTMC.

Recall that schedule is a set of links that are allowed to transmit at the same time, so a schedule has to be an independent set in the corresponding conflict graph. Further, define the transmission state of a link to be the indicator of whether the link is transmitting or not. We consider an algorithm that behaves as follows:

(i) The network first picks a set of link, say $\mathcal{D}$, which are allowed to change their transmission states. All other links are not allowed to change their transmission states. We assume that $\mathcal{D}$ itself is a valid schedule, although we do not use $\mathcal{D}$ as a schedule. We call set $\mathcal{D}$ a *decision schedule.*

(ii) Links in $\mathcal{D}$ that have a neighbor in the conflict graph which was transmitting in the previous slot will also not be allowed to change their transmission states.

(iii) Among the remaining links, each of them decides to turn on with probability α_l, for link l, and turn off with probability $(1-\alpha_l)$.

Note that $\mathcal{D}$ is a valid schedule, and links that can interfere with those links in the previous schedule were removed in step (ii), so after step (iii), the set of on links forms a valid schedule.

Let $\mathcal{S}(t)$ denote the schedule used in time slot t. Suppose $\mathcal{D}$ is chosen with some probability $p(\mathcal{D})$, independently at the beginning of each time slot. Then, $\mathcal{S}(t)$ is determined by $\mathcal{D}(t)$ and $\mathcal{S}(t-1)$ under the algorithm above, so forms a DTMC. If we are lucky, then the DTMC is reversible, and the steady state distribution is easy to verify.

We now derive conditions on $p(\mathcal{D})$ and α_l to ensure that the DTMC is reversible. For this purpose, let x be a schedule (independent set) and $x+m_1-m_2$ be another schedule, where m_1 are links that are not in x, and m_2 are links that are in x. Let us consider the transition from x to $x+m_1-m_2$ and vice versa.

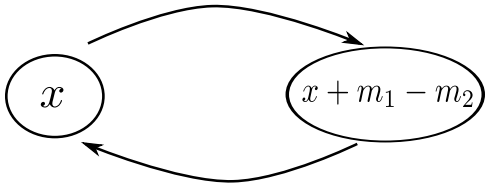

Figure 4.6: The transition from x to $x+m_1-m_2$ and vice versa

The transition from x to $x+m_1-m_2$ means the network will turn on those links in m_1 and turn off those links in m_2. Similarly, the transition from $x+m_1-m_2$ to x means the network will turn off those links in m_1 and turn on links in m_2. For the transition from x to $x+m_1-m_2$ to occur,

the decision schedule $\mathcal{D}$ must contain both m_1 and m_2 and similarly when in state $x + m_1 - m_2$. Pictorially, $\mathcal{D}$ will look like something in Figure 4.7.

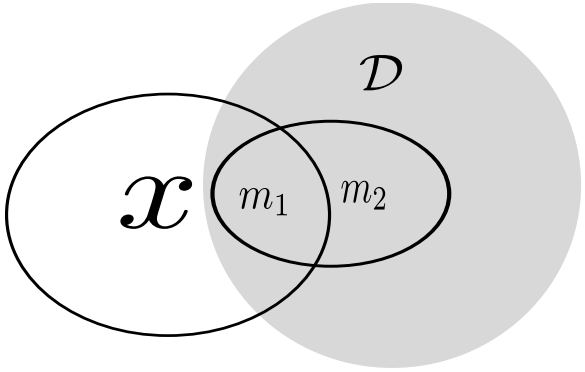

Figure 4.7: The gray area is $\mathcal{D}$ and $m_1 \cup m_2 \subseteq \mathcal{D}$. Other links can also belong to $\mathcal{D}$.

Let γ_x to the sum weight associated with schedule x, i.e., $\gamma_x = \sum_{l \in x} w_l$. According to Theorem 2.3.6, the DTMC has (4.15) as the stationary distribution if $\pi_x P_{x,y} = \pi_y P_{y,x}$, where x and y are two schedules. In other words, the DTMC has stationary distribution (4.15) when the following equation holds:

$$\frac{e^{\gamma_x}}{Z} P_{x,x+m_1-m_2} = \frac{e^{\gamma_x+\gamma_{m_1}-\gamma_{m_2}}}{Z} P_{x+m_1-m_2,x},$$

which is equivalent to

$$\begin{aligned} & \frac{e^{\gamma_x}}{Z} \sum_{\mathcal{D}: m_1 \cup m_2 \subseteq \mathcal{D}} \Pr(\mathcal{D} \backslash (m_1 \cup m_2) \text{ do not change state}) \\ & \qquad \times \Pr(m_1 \text{ turns on}) \times \Pr(m_2 \text{ turns off}) \times p(\mathcal{D}) \\ = \quad & \frac{e^{\gamma_x+\gamma_{m_1}-\gamma_{m_2}}}{Z} \sum_{\mathcal{D}: m_1 \cup m_2 \subseteq \mathcal{D}} \Pr(\mathcal{D} \backslash (m_1 \cup m_2) \text{ do not change state}) \\ & \qquad \times \Pr(m_1 \text{ turns off}) \times \Pr(m_2 \text{ turns on}) \times p(\mathcal{D}). \end{aligned}$$

A sufficient condition for the equality above to hold is

$$\begin{aligned} & e^{\gamma_{m_2}} \Pr(m_1 \text{ turns on}) \times \Pr(m_2 \text{ turns off}) \\ = \quad & e^{\gamma_{m_1}} \Pr(m_1 \text{ turns off}) \times \Pr(m_2 \text{ turns on}). \end{aligned} \tag{4.17}$$

Note that $\Pr(\mathcal{D} \backslash (m_1 \cup m_2)$ do not change state) is a complicated expression, but it is the same for each $\mathcal{D}$ on the right-hand-side and left-hand-side. It even depends on x, but this fact is irrelevant to us.

A sufficient condition for (4.17) to hold is for any subset of links m,

$$e^{\gamma_m} \Pr(m \text{ turns off}) = \Pr(m \text{ turns on}),$$

which is equivalent to

$$\prod_{l\in m} e^{w_l}(1-\alpha_l) = \prod_{l\in m} \alpha_l, \tag{4.18}$$

where α_l is the probability that link l turns on if it is part of the decision schedule and is allowed to change its transmission state. Matching the terms for the same l, we have that a sufficient condition for (4.18) to hold is

$$e^{w_l}(1-\alpha_l) = \alpha_l,$$

which requires

$$\alpha_l = \frac{e^{w_l}}{1+e^{w_l}}.$$

We note that α_l is a function of w_l only, and does not depend on the weights of other links, which means that link l can make on-off decision based on local information.

In summary, if link l knows it is in the decision schedule and none of its neighbors is in the previous schedule, then it can turn on-off probabilistically according to its weight w_l. The schedule $\mathcal{S}(t)$ a reversible Markov chain, and its steady-state distribution is (4.15).

We assume each link can sense whether other links in its neighborhood transmitted in the previous time-slot using carrier sensing, so both steps (ii) and (iii) can be done locally. So to have a distributed scheduling algorithm, we have to have a distributed mechanism to pick the decision set $\mathcal{D}$, which has to be a valid schedule so that each link in $\mathcal{D}$ can individually decide to be on or off without considering other links' decisions. Further, the process of generating $\mathcal{D}$ needs to guarantee that the DTMC is aperiodic and irreducible. The following protocol is one way to generate $\mathcal{D}$ in a distributed fashion:

(a) At the beginning of each time slot, each link transmits a "control" message with probability β, independent of all other links. The parameter β can be any arbitrary fixed value in $(0,1)$.

(b) If a control message collides with a neighbor's control message, then the link does not become a part of $\mathcal{D}$. Else, it become a part of decision set $\mathcal{D}$.

Clearly, $\mathcal{D}$ is an independent set because two conflicting links cannot both present in $\mathcal{D}$. Further, from any state x, the process above guarantees that the transition probability from x to the schedule in which all links are off is strictly positive, so the Markov chain is irreducible and aperiodic.

The Q-CSMA algorithm is presented in Algorithm 1.

Example 14 *Consider a three-link ad hoc network and its conflict graph as shown in Figure 4.8. Assume $x(t-1) = \{1,0,0\}$, i.e., link 1 is on and links 2 and 3 are off. In the following, we illustrate the Q-CSMA at time t for two specific decision schedules $\mathcal{D}_1 = \{1,3\}$ and $\mathcal{D} = \{2\}$:*

- *If the decision schedule is $\{1,3\}$, link 1 will continue to transmit with probability $\frac{e^{w_1}}{1+e^{w_1}} = \frac{e}{1+e}$, and turns off with probability $\frac{1}{1+e}$, and link 3 turns on with probability $\frac{e^{w_3}}{1+e^{w_3}} = \frac{e}{1+e}$ and turns off with probability $\frac{1}{1+e}$.*
- *If the decision schedule is $\{2\}$, since link 1 was transmitting in the previous time slot, link 2 does not change its transmission state and will remain off.*

□

Algorithm 1 Q-CSMA

while $t \geq 0$ **do**
  Set $w_l = \log(1 + q_l(t))$.
  At the beginning of each time slot, each link transmits a "control" message with probability β, independent of all other links. The parameter β can be any arbitrary fixed value in $(0, 1)$.
  if A control message collides with a neighbor's control message. **then**
    The link does not become a part of $\mathcal{D}$.
  else
    The link become a part of decision set $\mathcal{D}$.
  end if
  for $l \in \mathcal{D}$ **do**
    if link l has a neighbor in the conflict graph which was transmitting in the previous time slot. **then**
      link l does not change its transmission state.
    else
      link l turns on with probability $\frac{e^{w_l}}{1+e^{w_l}}$, and turns off with probability $\frac{1}{1+e^{w_l}}$.
    end if
  end for
end while

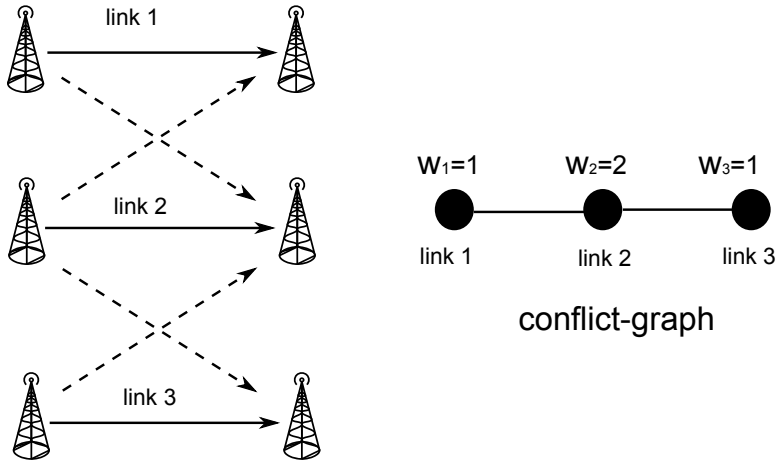

Figure 4.8: A three-link example illustrating the Q-CSMA

We finally comment that we have made a number of assumptions in describing the above algorithm:

- Each link can sense whether other links in its neighborhood transmitted in the previous time-slot. This is accomplished by a protocol called "carrier-sensing". Each link in our model senses the presence or absence of transmission energy or a "carrier" its neighborhood. Hence, the algorithm is Carrier Sensing Multiple Access.

- We also assume that when two control messages collide, it can be detected.

- We assume that the process of selecting $\mathcal{D}$ takes negligible time. This is a reasonable assumption if packet sizes are much longer than the time to transmit an intent message and detect collisions.
- We assume time is slotted.

Some of these assumptions will not hold in practice. At the end of the chapter, we will provide references which address these issues.

Chapter 5

Back to Network Utility Maximization

In the previous chapter, we looked at scheduling in wireless networks, where the focus was to design a scheduling algorithm that can support any traffic strictly within the capacity region. Implicitly, we assumed that data flows were regulated by congestion control so the incoming traffic was always within the capacity region. However, in practice, the behaviors of congestion control algorithms themselves may be affected by scheduling algorithms. For example, if TCP-Vegas were implemented, then the source rates would be regulated based on queueing delays, which could be quite different under different scheduling algorithms.

In this chapter, we will consider the network utility maximization problem for wireless networks, and derive joint congestion control, routing and scheduling algorithms that maximize the net utility. For wireless networks, the link capacity constraints will be written differently from those in wired networks. The rationale behind this difference will be explained at the end of this chapter.

5.1 A Joint Formulation of the Transport, Network and MAC Problems

We consider a wireless network with N nodes that are distributed in a two-dimensional space. We are going to take into account both interference and channel fading in the same model. The transmission rates between node pairs are determined by the channel state in the network, denoted by m. Recall that the channel state is a variable which describes the channel conditions in the network. For example, if the channel conditions are good, then the realized data rates will be high; otherwise, the rates will be low.

In addition to the channel state, interference determines the data rates at which different node pairs can communicate at the same time. Let r denote the data rate vector, where r_{ij} is the data rate from node i to node j. The vector r is assumed to take on values in a discrete set $\mathcal{R}_m$, when the channel is in state m. The set $\mathcal{R}_m$ captures interference effects and power constraints. We further assume that given any $r \in \mathcal{R}_m$, $r^{(i)}$, such that $r_i^{(i)} = 0$ and $r_j^{(i)} = r_j$ for $j \neq i$, also belongs to $\mathcal{R}_m$. Under this reasonable assumption, $Co(\mathcal{R}_m)$ is coordinate convex. The following example illustrates the idea.

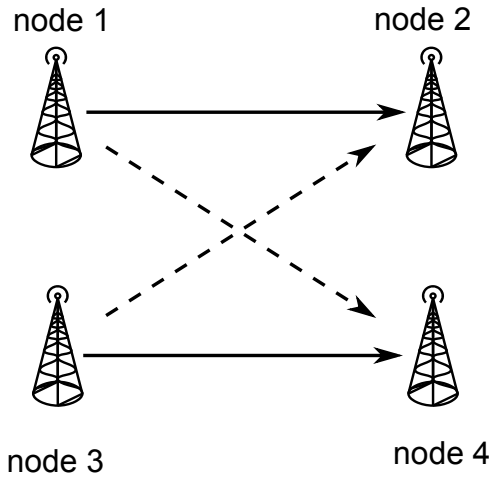

Figure 5.1: A four-node wireless network. The solid lines represent intended transmissions and dashed lines represent interference.

Example 15 *Consider a four-node network as shown in Figure 5.1. Node 1 wants to transmit to node 2, and node 3 wants to transmit to node 4. For simplicity, assume each node has three different power levels:* $\{0, p_{1a}, p_{1b}\}$ *are the power levels available at node 1 and* $\{0, p_{3a}, p_{3b}\}$ *are the power levels available at node 3. Further suppose that* h_{ij} *is the channel gain from node* i *to node* j*, i.e., if node* i *transmits at power* P_i*, then the received power at node* j *is* $P_i h_{ij}$*. The transmission rates are functions of SINRs, i.e.,*

$$r_{12} = f(P_1, P_3, h_{12}, h_{32}, n_2)$$
$$r_{34} = f(P_3, P_1, h_{34}, h_{14}, n_4),$$

where n_i *is the variance of the Gaussian background noise experienced by node* i *and* f *is some function. By varying the power levels* P_1 *and* P_3 *over* $\{0, p_{1a}, p_{1b}\}$ *and* $\{0, p_{3a}, p_{3b}\}$*, respectively, we get different values of* (r_{12}, r_{34})*, which is what we denote by the set* $\mathcal{R}_m$*. In this example, the channel state can be thought of as depicting the channel gains* $\{h_{ij}\}$*. We assume that if* $r \in \mathcal{R}_m$*, then the same vector with one element replaced by zero also belongs to* $\mathcal{R}_m$*. In other words, a user has the choice to transmit at rate zero.*

Assuming simple coding and modulation schemes, the transmission rates can be written as

$$r_{12} = \frac{1}{2}\log_2\left(1 + \frac{P_1 h_{12}}{n_2 + P_3 h_{32}}\right)$$
$$r_{34} = \frac{1}{2}\log_2\left(1 + \frac{P_3 h_{34}}{n_4 + P_1 h_{14}}\right).$$

Assuming the channel gains, power levels and variances of the noises are as shown in Table 5.1, the set $\mathcal{R}_m$ *is illustrated in Figure 5.2. Note a vector in* $Co(\mathcal{R}_m)$ *is a linear combination of vectors in* $\mathcal{R}_m$*, so any rate vector in* $Co(\mathcal{R}_m)$ *can be achieved by time-sharing. The capacity region given channel state* m*, therefore, is* $Co(\mathcal{R}_m)$*. Given the parameters in Table 5.1, the capacity region is shown in Figure 5.3.* □

Channel gain	$h_{12} = h_{34} = 1$ and $h_{14} = h_{32} = 0.1$
Power levels	$P_1 \in \{0, 10, 15\}$ and $P_3 \in \{0, 5, 10\}$
Variances of noises	$n_2 = n_4 = 1$.

Table 5.1: Channel gains, power levels, and variances of noises

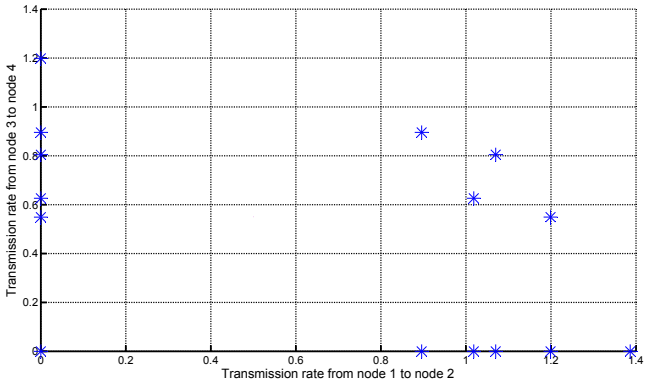

Figure 5.2: The set $\mathcal{R}_m$.

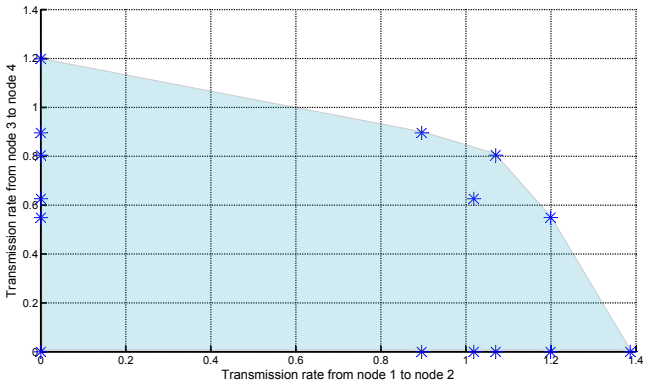

Figure 5.3: The set $Co(\mathcal{R}_m)$.

In general, the channel state can vary over time. In our discrete-time model, we assume that π_m is the fraction of time that the channel is in state m. Assume there are M possible states, then

$$\sum_{1 \le m \le M} \pi_m = 1.$$

Now let r be an element of the set $\mathcal{R}_m$, and $\alpha_{m,r}$ be the fraction of time that the rate vector r is used by the network when the channel state is m. Then the data rate realized from node i to

node j is given by

$$\mu_{ij} = \sum_m \pi_m \sum_{r \in \mathcal{R}_m} \alpha_{m,r} r_{ij}.$$

The set of achievable rates (achieved by varying α) is called the link capacity region, and is denoted by $\mathcal{C}$.

We note that for given m,

$$Co(\mathcal{R}_m) = \left\{ \mu : \mu = \sum_{r \in \mathcal{R}_m} \alpha_{m,r} r \text{ such that } \alpha_m \geq 0 \text{ and } \sum_{r \in \mathcal{R}_m} \alpha_{m,r} = 1 \right\}.$$

Therefore, an equivalent way to express $\mathcal{C}$ is

$$\mathcal{C} = \left\{ \mu : \mu = \sum_m \pi_m r_m, \quad r_m \in Co(\mathcal{R}_m) \right\}.$$

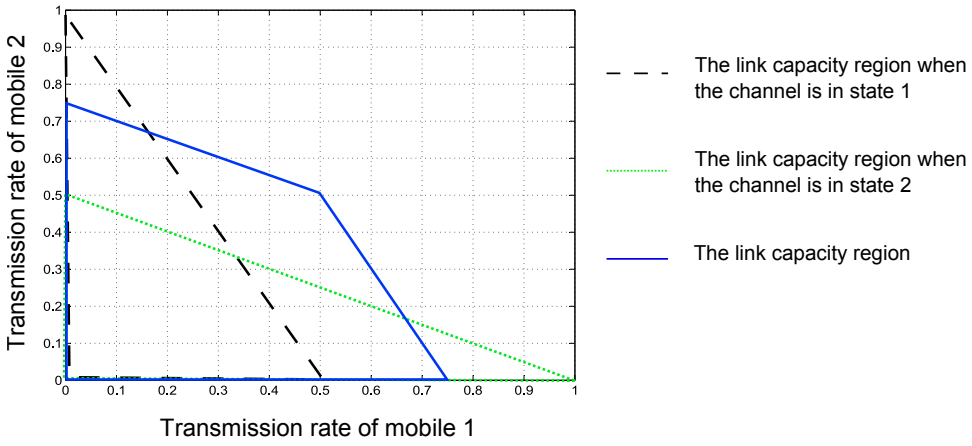

Figure 5.4: The link capacity region with fading channels.

Example 16 *Consider a network with two links. Assume the network has two states: state 1 and state 2. Each state occurs with probability* 1/2. *The link capacity region is plotted in Figure 5.4.* □

We now introduce the network utility maximization problem for general wireless networks, where multihop flows are allowed. Let $\mathcal{F}$ be the set of flows in the network. A flow f is associated with a source node $s(f)$ and a destination node $d(f)$. We assume that a flow can split its packets along multiple routes. In our model, even the routes are not pre-specified, in general. The algorithm that we will design will automatically find appropriate routes through the network.

Let x_f be the rate at which source $s(f)$ generates data. Let $\mu_{ij}^{(d)}$ be the rate at which the data to destination d is transmitted over link (i,j). For $\{x_f\}$ to be accommodated in the network, $\{\mu_{ij}^{(d)}\}$ must satisfy the following equation for each node i and each destination d :

$$x_f \mathbb{I}_{s(f)=i, d(f)=d} + \sum_{j:(j,i) \in \mathcal{L}} \mu_{ji}^{(d)} \leq \sum_{k:(i,k) \in \mathcal{L}} \mu_{ik}^{(d)}, \tag{5.1}$$

where $\mathcal{L}$ is the set of links. This constraint says that, at each node i, the total incoming data rate due to destination d (the left-hand-side of the constraint) must be less than or equal to the total outgoing rate allocated to destination d (the right-hand-side of the constraint).

Associate a concave utility function $U_f(x_f)$ with flow f. Our resource allocation problem is to find the appropriate α to solve the following network utility maximization problem.

$$\max_{x,\alpha\geq 0} \sum_f U_f(x_f) \tag{5.2}$$

$$\text{subject to: } x_f \mathbb{I}_{s(f)=i,d(f)=d} + \sum_j \mu_{ji}^{(d)} \leq \sum_k \mu_{ik}^{(d)} \quad \forall i \quad \forall d \tag{5.3}$$

$$\sum_d \mu_{ij}^{(d)} = \sum_m \pi_m \sum_{r\in\mathcal{R}_m} \alpha_{m,r} r_{ij} \quad \forall (i,j) \in \mathcal{L} \tag{5.4}$$

$$\sum_{r\in\mathcal{R}_m} \alpha_{m,r} = 1 \quad \forall m \tag{5.5}$$

By abusing notation for convenience, we will denote constraints (5.4) and (5.5) by

$$\mu \in \mathcal{C}.$$

To solve the problem, append (5.3) to the objective using Lagrange multipliers to get

$$\max_{x,\mu\geq 0} \sum_f U_f(x_f) - \sum_{i,d} \phi_{id} \left(x_f \mathbb{I}_{s(f)=i,d(f)=d} + \sum_j \mu_{ji}^{(d)} - \sum_k \mu_{ik}^{(d)} \right)$$

subject to

$$\mu \in \mathcal{C}.$$

Note that for fixed ϕ_{id}, the maximization over x and μ can be separated. The optimization problem can be decomposed into the following two optimization problems:

$$\textbf{Sub-problem 1:} \quad \sum_f \max_{x_f\geq 0} \left(U_f(x_f) - \phi_{s(f)d(f)} x_f \right), \tag{5.6}$$

and

$$\textbf{Sub-problem 2:} \quad \max_{\mu\in\mathcal{C}} \sum_{i,d} \phi_{id} \left(\sum_k \mu_{ik}^{(d)} - \sum_j \mu_{ji}^{(d)} \right) \tag{5.7}$$

If the Lagrange multipliers ϕ are known, then the sub-problem 1 (5.6) is equivalent to

$$\max_{x_f\geq 0} U_f(x_f) - \phi_{s(f)d(f)} x_f \quad \forall f. \tag{5.8}$$

Thus, each user can determine its transmission rate based on the Lagrange multiplier associated with its ingress node and its destination.

To understand (5.7), note that

$$\sum_{i,d} \phi_{id} \sum_k \mu_{ik}^{(d)} = \sum_{i,k} \sum_d \mu_{ik}^{(d)} \phi_{id} = \sum_{i,j} \sum_d \mu_{ij}^{(d)} \phi_{id}$$

and

$$\sum_{i,d} \phi_{id} \sum_j \mu_{ji}^{(d)} = \sum_{i,j} \sum_d \mu_{ji}^{(d)} \phi_{id} = \sum_{i,j} \sum_d \mu_{ij}^{(d)} \phi_{jd},$$

where the second equality is obtained by switching the indices i and j. Therefore, (5.7) can be written as

$$\max_{\mu \in \mathcal{C}} \sum_{i,j} \sum_d \mu_{ij}^{(d)} (\phi_{id} - \phi_{jd}) = \max_{\mu \in \mathcal{C}} \sum_{ij} \max_d (\phi_{id} - \phi_{jd}) \left(\sum_d \mu_{ij}^{(d)} \right).$$

Substituting for $\mu_{ij}^{(d)}$ from (5.4) yields

$$\max_{\alpha} \sum_{i,j} \max_d (\phi_{id} - \phi_{jd}) \left(\sum_m \pi_m \sum_{r \in \mathcal{R}_m} \alpha_{m,r} r_{ij} \right).$$

Using the linearity of the above expression, it can be simplified to (left as an exercise to the reader):

$$\max_{r \in \mathcal{R}_m} \sum_{i,j} r_{ij} \max_d (\phi_{id} - \phi_{jd}). \tag{5.9}$$

Note that (5.9) is the so-called *back-pressure algorithm.* It is a MaxWeight algorithm, where the weight of each link is

$$\max_d (\phi_{id} - \phi_{jd}),$$

and $(\phi_{id} - \phi_{jd})$ is called the *backpressure* of destination d on link (i, j). Note that if

$$\max_d (\phi_{id} - \phi_{jd}) < 0$$

then the MaxWeight algorithm will choose $r_{ij} = 0$.

If the Lagrange multipliers $\{\phi_{id}\}$ were known, then the congestion control algorithm (5.8) and the MaxWeight scheduling algorithm (5.9) solve the resource allocation problem. So the problem becomes to estimate the Lagrange multipliers.

As in the dual algorithm for the Internet, a natural algorithm to estimate ϕ_{id} is the following:

$$\phi_{id}(t+1) = \left(\phi_{id}(t) + \epsilon \left(x_f(t) \mathbb{I}_{s(f)=i, d(f)=d} + \sum_j \mu_{ji}^{(d)}(t) - \sum_k \mu_{ik}^{(d)}(t) \right) \right)^+, \tag{5.10}$$

where $\mu_{ji}^{(d)}(t)$ and $x_f(t)$ are the solutions of (5.6) and (5.7) with ϕ_{id} replaced by the estimate $\phi_{id}(t)$. Note that (5.10) is a difference equation counterpart of the differential equation in the dual algorithm for the Internet, and ϵ is a step-size parameter.

Computing $\mu_{ji}^{(d)}(t)$ requires the knowledge of π_m, which may be unknown. Suppose that the channel state at time t is $m(t)$, and $r(t)$ is the solution to (5.9), then we can try to use the following difference equation to estimate the Lagrange multipliers:

$$\phi_{id}(t+1) = \left[\phi_{id}(t) + \epsilon \left(x_f(t) \mathbb{I}_{s(f)=i, d(f)=d} + \sum_j r_{ji}^{(d)}(t) - \sum_k r_{ik}^{(d)}(t) \right) \right]^+, \tag{5.11}$$

where

$$r_{ij}^{(d)}(t) = \begin{cases} r_{ij}(t), & \text{if } d = d^* \\ 0, & \text{otherwise.} \end{cases} \tag{5.12}$$

and d^* is a solution to

$$\max_d (\phi_{id}(t) - \phi_{jd}(t)).$$

If more than one d achieves the maximum, then an arbitrary such d is chosen to be d^*.

The behavior of ϕ_{id}/ϵ in equation (5.11) looks almost like the queue length at node i for destination d. The only difference from the real queue dynamics is that it may not be possible to transfer $r_{ji}^{(d)}(t)$ packets from q_{jd} to q_{id}[1] since there may not be enough packets in q_{jd}. Thus, the true queue dynamics would satisfy

$$q_{id}(t+1) \leq \left[q_{id}(t) + \left(x_f(t)\mathbb{I}_{s(f)=i,d(f)=d} + \sum_j r_{ji}^{(d)}(t) - \sum_k r_{ik}^{(d)}(t) \right) \right]^+. \tag{5.13}$$

The exact dynamics would depend on the policy used to allocate packets along multiple links at each node when there are not enough packets to transfer as dictated by the MaxWeight algorithm. As we will see, the exact dynamics are not important for our analysis. Using the queue length q_{id} to approximate ϕ_{id} turns out to be good enough.

Based on the derivation above, we obtain a joint congestion control, routing and scheduling algorithm, where each source adjusts its transmission rate based on optimization problem (5.8), and the network determines routing and scheduling by solving optimization problem (5.9). The Lagrange multipliers in (5.8) and (5.9) are replaced by per-destination queues maintained at each node. A detailed description is presented in Algorithm 2.

Note that in (5.14), we added a constraint $x_f \leq x_{\max}$ so that the source will not transmit with infinite speed when $q_{s(f),d(f)}(t) = 0$. This constraint does not change the optimal solution when $x_{\max} > x_f^*$, which can be guaranteed by choosing a sufficiently large $x_{\max}$, e.g., $x_{\max} \geq r_{\max}$, where $r_{\max}$ is the maximum link capacity in the network.

Example 17 *Consider a network with three nodes as shown in Figure 5.5, where we have two flows: one flow from node 1 to node 4, and another flow from node 1 to node 2. In this network, all nodes maintain a queue for node 4 because node 4 is reachable from all other nodes in the network. Only nodes 1 and 2 maintain queues for node 2 because node 2 cannot be reached from other nodes. We further note that* $q_{dd} \equiv 0$, *so* $q_{22} = q_{44} = 0$.

Assume that the utility function of flow $(1,4)$ *is* $2\log x_{(1,4)}$, *and the utility function of flow* $(1,2)$ *is* $\log x_{(1,2)}$. *Further assume and* $\epsilon = 0.1$, $x_{\max} = 10$, *and* $\mathcal{R}_{m(t)} = \{(r_{1,2}, r_{1,3}, r_{2,4}, r_{3,4}) : (4,0,0,5), (0,2,3,0)\}$. *Then Algorithm 2 works as follows:*

- *Given the utility function, we can explicitly solve (5.14). We have*

$$x_{(1,4)}(t) = \min\left\{ \frac{2}{\epsilon q_{14}(t)}, x_{\max} \right\} = 2$$

$$x_{(1,2)}(t) = \min\left\{ \frac{1}{\epsilon q_{12}(t)}, x_{\max} \right\} = 1.$$

[1] Abusing terminology, we use q_{jd} to denote the queue at node i for destination d and also use it to denote the corresponding queue length.

Algorithm 2 Joint Congestion Control, Routing, and Scheduling Algorithm

1: **while** $t \geq 0$ **do**

2: **Congestion control:** The base station computes $x_f(t)$ by solving the following optimization problem:

$$\max_{0 \leq x_f \leq x_{\max}} U_f(x_f) - \epsilon q_{s(f)d(f)}(t) x_f, \tag{5.14}$$

and then generates $a_f(t)$ packets, where $a_f(t)$ is a random variable taking integer values and with mean $x_f(t)$ (e.g., $a_f(t)$ can be a Poisson random variable with parameter $x_f(t)$).

3: **Routing:** Node i routes packets with destination $d_{ij}^*(t)$ from node i to node j, where

$$d_{ij}^*(t) \in \arg\max_d (q_{id}(t) - q_{jd}(t)).$$

4: **Rate scheduling:** The network schedules link rate vector $r(t)$ such that

$$r(t) \in \arg\max_{r \in \mathcal{R}_{m(t)}} \sum_{i,j} w_{ij}(t) r_{ij},$$

where $w_{ij}(t) = q_{id_{ij}^*(t)}(t) - q_{jd_{ij}^*(t)}(t)$.

5: **Data transmission:** Node i transmits $\mu_{ij}(t)$ packets with destination d_{ij}^* to node j, which will be stored in queue $q_{jd_{ij}^*}$, where $\mu_{ij}(t) = \min\{r_{ij}(t), q_{id_{ij}^*(t)}(t)\}$.

6: **Queue dynamics:** The queue q_{id} evolves as follows:

$$q_{id}(t+1) = q_{id}(t) + a_f(t)\mathbb{I}_{s(f)=i, d(f)=d} + \sum_j \mu_{ji}^{(d)}(t) - \sum_k \mu_{ik}^{(d)}(t).$$

7: **end while**

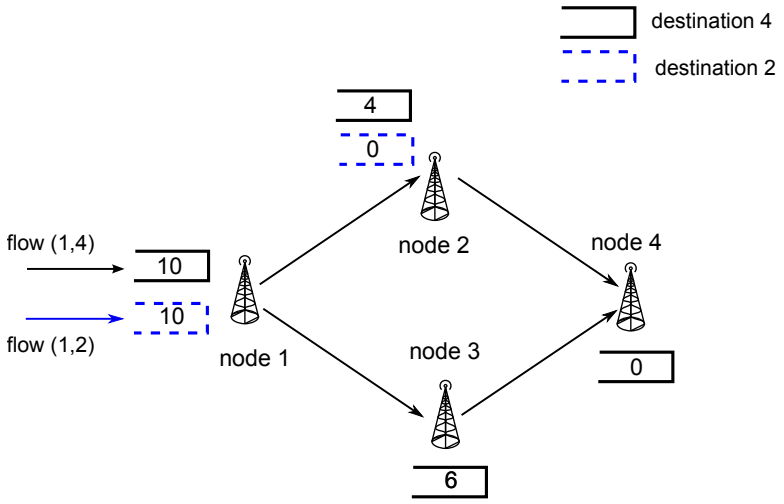

Figure 5.5: A four-node wireless network.

- *At link* $(1,2)$*, the backpressure of destination 4 is 6 and the backpressure of destination 2 is 10, so link* $(1,2)$ *will transmit packets to destination 2, and the link weight* $w_{12} = 10$*. All other links serve packets to destination 4, and the link weights are* $w_{13} = 4$*,* $w_{34} = 6$ *and* $w_{24} = 4$*.*

- *Given the link weights, it is not hard to verify that link rate vector* $\{4,0,0,5\}$ *results in a larger* $\sum_{i,j} w_{ij} r_{ij}$ *than* $\{0,2,3,0\}$*. Therefore, node 1 transmits 4 packets with destination 2 to node 2, and node 3 transmits 5 packets with destination 4 to node 4.*

□

To understand the performance of the joint algorithm, we will assume that $\{a_f(t)\}$ are independent across time, and the channel state process is i.i.d. across time with π_m being the probability the channel is in state m. Under these conditions, $q(t)$ is a DTMC. We further assume that the arrival processes and channel state process are such that the DTMC is irreducible. We then have the following theorem, which states that under the joint algorithm, the expected sum queue length is bounded and the resource allocation maximizes the network utility when $\epsilon \to 0$.

Theorem 5.1.1 *The irreducible DTMC* $q(t)$ *is positive recurrent under the joint algorithm and*

$$\lim_{t\to\infty} E\left[\sum_{i,d} q_{id}(t)\right] \leq \frac{B_1}{\epsilon} \tag{5.15}$$

for some $B_1 < \infty$*.*

Further, let

$$y_f = \lim_{t\to\infty} E\left[x_f(t)\right].$$

Then

$$\sum_f U_f(x_f^*) - B_2\epsilon \leq \sum_f U_f(y_f) \tag{5.16}$$

for some $B_2 < \infty$*, where* $\{x_f^*\}$ *is the solution to* (5.2) *subject to* (5.3)*,* (5.4) *and* (5.5)*.* □

Note that the theorem suggests a tradeoff: if our realized utility $\sum_f U_f(y_f)$ is close to the optimal utility $\sum_f U_f(x_f^*)$, then the upper bound on the sum of the queue lengths is large $O(\frac{1}{\epsilon})$. While the estimate of the sum queue lengths is only an upper bound, this behavior is to be expected for the following reason: if we get close to the optimal solution, it means that some of the constraints (5.3) must be close to tight. But these constraints simply state that the arrival rate is no more than the departure rate at a queue. Recalling our discussion of discrete time queuing systems earlier, when the arrival rate is close to the service rate, the expected queue length in the steady-state can be large.

We will not prove the above theorem. We will discuss some special cases and provide a proof in one of these special cases.

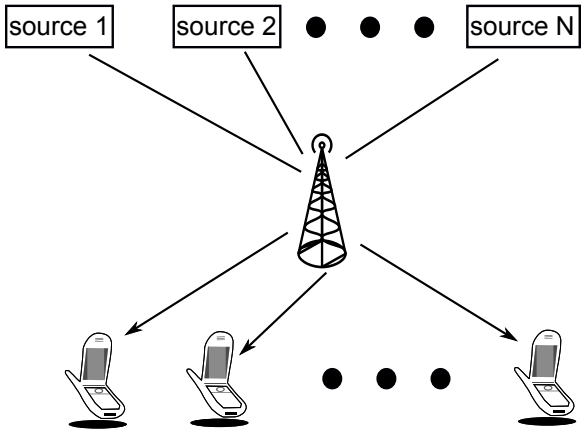

Figure 5.6: A cellular network with N mobiles

5.2 Stability and Convergence: An Example for Cellular Networks

We first consider a cellular downlink network as shown in Figure 5.6, where packets are injected by the sources into the base station, and the base station needs to deliver the packets to the mobiles. Here since each flow goes over only one link (from the base station to a mobile), the notation can be simplified as follows:

- x_i: the average rate at which the source sending content to mobile i injects data into the base station,
- μ_i: the average rate at which the base station transmits to mobile i.

The rest of the variables are as before.

The NUM problem in this case is

$$\begin{aligned} & \max_{x,\alpha,\mu\geq 0} \sum_{i=1}^{N} U_i(x_i) \\ \text{subject to} \quad & x_i \leq \mu_i \\ & \mu_i = \sum_m \pi_m \left(\sum_{r\in\mathcal{R}_m} \alpha_{m,r} r_i \right). \end{aligned}$$

Using the decomposition as before, we arrive at Algorithm 3.

In the above, we make the same assumptions as before on arrival processes and channel-state distribution so that $q(t)$ is an irreducible Markov chain. We will give a proof of Theorem 5.1.1 for this special case.

Proof In this proof, we will first show $q(t)$ is positive recurrent, and then prove the resource allocation under the joint algorithm is close to the optimal resource allocation based on the fact that $q(t)$ is positive recurrent. Consider the Lyapunov function

$$V(q(t)) = \frac{1}{2} \sum_i q_i^2(t).$$

Algorithm 3 Joint Congestion Control and Scheduling for Cellular Downlink Networks

1: **while** $t \geq 0$ **do**

2: **Congestion control:** Source i injects $a_i(t)$ packets into the base station such that $E[a_i(t)] = x_i(t)$, and

$$x_i(t) \in \arg \max_{0 \leq x_i \leq x_{\max}} U_i(x_i) - \epsilon q_i(t) x_i.$$

3: **MaxWeight scheduling:** The transmission rate vector $r(t)$ is chosen to be a solution of the following optimization problem:

$$\max_{r \in \mathcal{R}_{m(t)}} \sum_{i=1}^{N} r_i q_i(t).$$

Ties are broken at random.

4: **The evolution of the queues:** The queue maintained for mobile i evolves as follows:

$$q_i(t+1) = \left(q_i(t) + a_i(t) - r_i(t)\right)^+.$$

5: **end while**

The drift of the Lyapunov function given the queue and channel states is

$$\begin{aligned}
& E\left[V(q(t+1)) - V(q(t))|q(t) = q\right] \\
= \; & \frac{1}{2} E\left[\sum_i q_i^2(t+1) - q_i^2(t) \middle| q(t) = q\right] \\
= \; & \frac{1}{2} E\left[\sum_i ((q_i(t) + a_i(t) - r_i(t))^+)^2 - q_i^2(t) \middle| q(t) = q\right] \\
\leq \; & \frac{1}{2} E\left[\sum_i (q_i(t) + a_i(t) - r_i(t))^2 - q_i^2(t) \middle| q(t) = q\right] \\
= \; & \frac{1}{2} E\left[\sum_i (a_i(t) - r_i(t))^2 + 2q_i(t)(a_i(t) - r_i(t)) \middle| q(t) = q\right] \\
\leq \; & K + \sum_i q_i x_i(t) - \sum_i q_i E[r_i(t)|q(t) = q],
\end{aligned}$$

where we assume $E[a_i^2(t)] \leq \sigma_{\max}^2$ and $K = \frac{N\left(\sigma_{\max}^2 + r_{\max}^2\right)}{2}$. Now adding and subtracting $\frac{1}{\epsilon}\sum_i U_i(x_i)$ at the right-hand-side of the inequality above results in

$$\begin{aligned}
& E\left[V(q(t+1)) - V(q(t))|q(t) = q\right] \\
\leq \; & K + \sum_i q_i x_i(t) - \frac{1}{\epsilon}\sum_i U_i(x_i(t)) + \frac{1}{\epsilon}\sum_i U_i(x_i(t)) - \sum_i q_i \bar{r}_i,
\end{aligned}$$

where $\bar{r}_i = E[r_i(t)|q(t) = q]$.

Since $x_i(t)$ maximizes $U_i(x) - \epsilon q_i x$ for $0 \leq x \leq x_{\max}$, and $x_i^* \leq x_{\max}$ (recall x^* is the optimal

resource allocation), we have

$$\frac{1}{\epsilon}U_i(x_i^*(1-\delta)) - q_i x_i^*(1-\delta) \le \frac{1}{\epsilon}U_i(x_i(t)) - q_i x_i(t),$$

where $0 \le \delta < 1$. So we can obtain

$$\begin{aligned} & E\left[V(q(t+1)) - V(q(t))|q(t)=q\right] \\ \le\ & K + \sum_i q_i x_i^*(1-\delta) - \sum_i \frac{1}{\epsilon}U_i(x_i^*(1-\delta)) + \frac{1}{\epsilon}\sum_i U_i(x_i(t)) - \sum_i q_i \bar{r}_i \\ =\ & K + \sum_i q_i(x_i^* - \bar{r}_i) - \delta\sum_i q_i x_i^* + \frac{1}{\epsilon}\sum_i (U_i(x_i(t)) - U_i(x_i^*(1-\delta))) \qquad (5.17) \\ \le\ & \tilde{K} + \sum_i q_i(x_i^* - \bar{r}_i) - \delta\sum_i q_i x_i^*, \end{aligned}$$

where $\tilde{K} = K + \frac{1}{\epsilon}\sum_i (U_i(x_{\max}) - U_i(x_i^*(1-\delta)))$. It is not difficult to show that under MaxWeight scheduling, $r(t)$ solves

$$\max_{r\in\mathcal{R}_{m(t)}} \sum_i r_{m(t),i} q_i,$$

and $\bar{r} = E[r(t)|q(t)=q]$ is a maximizer to the following problem:

$$\max_{r\in\mathcal{C}} \sum_i r_i q_i.$$

So $\sum_i q_i x_i^* \le \sum_i q_i \bar{r}_i$ and

$$E\left[V(q(t+1)) - V(q(t))|q(t)=q\right] \le \tilde{K} - \delta\sum_i q_i x_i^*. \qquad (5.18)$$

Since δ can be chosen to any value between 0 and 1, $q(t)$ is positive positive recurrence according to the Foster-Lyapunov theorem if $x_i^* > 0$ for all i.

Thus, we do not even need the strict concavity of $U_i(x_i)$. We only need concavity and the requirement that there exists an optimal solution x^* such that $x_i^* > 0$ for all i, which we assume to be true.

Since $q(t)$ is positive recurrent, a stationary distribution for $q(t)$ exists, and

$$E\left[V(q(t+1)) - V(q(t))\right] = 0$$

in the steady-state. So assuming $q(t)$ is in the steady state, we take expectations at both sides of inequality (5.18) and obtain

$$0 \le \tilde{K} - \delta\sum_i E\left[q_i(t)\right] x_i^*,$$

which implies that

$$E\left[\sum_i q_i(t)\right] \le \frac{\tilde{K}}{\delta \min_i x_i^*}$$

when $q(t)$ is in the steady-state.

Since $\sum_i q_i x_i^* \leq \sum_i q_i \bar{r}_i$, by choosing $\delta = 0$, we can further obtain from (5.17) that

$$E\left[\sum_i U_i(x_i^*) - U_i(x_i(t))\right] \leq \epsilon K,$$

or

$$E\left[\sum_i U_i(x_i^*)\right] \leq E\left[\sum_i U_i(x_i(t))\right] + \epsilon K$$

when $q(t)$ is in the steady state.

We remark that that the upper bound on the sum queue length is at the order of $O(1/\epsilon)$ while the difference between the network utility under the joint algorithm and the optimal network utility is at the order of $O(\epsilon)$.

□

5.3 Ad Hoc P2P Wireless Networks

We now consider network utility maximization in ad hoc P2P network. We assume that the channel state does not vary so we omit the subscript m. Further, instead of using a double subscript (i, j) to denote a link, we use l to denote a link. We let x_l denote the rate at which the source using link l transmits data, and μ_l denote the rate allocated to link l by the network. The necessary conditions for x_l to be supportable include

$$x_l \leq \mu_l \quad \forall l \tag{5.19}$$

$$\mu_l = \sum_{r \in \mathcal{C}} \alpha_r r_l \quad \forall l \tag{5.20}$$

$$\sum_{r \in \mathcal{R}} \alpha_r = 1. \tag{5.21}$$

The NUM problem is therefore to

$$\max_{x,\mu,\alpha \geq 0} \sum_l U_l(x_l)$$

subject to the constraints in (5.19)-(5.21).

The joint congestion control and scheduling algorithm is presented in Algorithm 4. The proof is similar to the cellular case, and left as an exercise.

Often, the interference constraints in the ad hoc network can be represented by a conflict graph as we explained in the previous chapter, which is a special case of the model we described here. To see this, we will consider a simple example.

Example 18 *Consider the conflict graph as shown in Figure 5.7. In this example, link 1 and link 4 can be scheduled simultaneously or link 2 and link 3 can be scheduled simultaneously. Assume*

Algorithm 4 Joint Congestion and Scheduling for Ad Hoc Peer-to-Peer Networks

1: **while** $t \geq 0$ **do**

2: **Congestion control:** The transmitter of link l computes the rate

$$x_l(t) = \arg \max_{0 \leq x_l \leq x_{\max}} U_l(x_l) - \epsilon q_l(t) x_l,$$

and injects $a_l(t)$ packets into its buffer, where $a_l(t)$ is an integer and $E[a_l(t)] = x_l(t)$.

3: **MaxWeight scheduling:** The network selects the link rate vector $r(t)$ such that

$$r(t) \in \arg \max_{r \in \mathcal{R}} \sum_l q_l(t) r_l.$$

Ties are broken at random.

4: **Queue evoluation:** The queue maintained at the transmitter of link l evolves as follows:

$$q_l(t+1) = \left(q_l(t) + a_l(t) - r_l(t)\right)^+.$$

5: **end while**

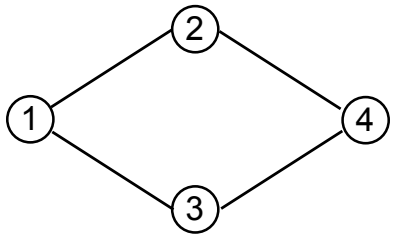

Figure 5.7: An example of conflict graph, in which a node represents a wireless link and a link represents a conflict of two links.

that when link l is scheduled, it can transmit at a maximum rate of c_l, which we will assume is an integer. Then, the possible rate vectors under the two schedules are:

$$a = \begin{pmatrix} c_1 \\ 0 \\ 0 \\ c_4 \end{pmatrix} \quad \textit{and} \quad b = \begin{pmatrix} 0 \\ c_2 \\ c_3 \\ 0 \end{pmatrix}$$

The convex hull of these two rate vectors and vector 0 is given by

$$Co(\{a, b, 0\}) = \{r : r = \alpha a + \beta b \textit{ for some } \alpha, \beta \geq 0 \textit{ and } \alpha + \beta \leq 1\},$$

which is also the capacity region obtained based on the conflict graph and convex hull model described in the previous chapter.

Instead, one can also enumerate all possible rate vectors as follows:

$$\mathcal{A} = \left\{ \begin{pmatrix} r_1 \\ 0 \\ 0 \\ r_4 \end{pmatrix} : r_1 \le c_1, r_4 \le c_4, r_1, r_4 \ge 0, \textit{integers} \right\}$$

and

$$\mathcal{B} = \left\{ \begin{pmatrix} 0 \\ r_2 \\ r_3 \\ 0 \end{pmatrix} : r_2 \le c_2, r_3 \le c_3, r_2, r_3 \ge 0, \textit{integers} \right\}.$$

Letting $\hat{\mathcal{C}} = \mathcal{A} \cup \mathcal{B}$, then $\mathcal{C}$ can be equivalently defined as

$$\mathcal{C} = \{r : \exists \gamma_i \in \hat{\mathcal{C}}, \alpha_i \ge 0, \sum_i \alpha_i = 1 \textit{ such that } r = \sum_i \alpha_i \gamma_i\}.$$

This is the definition that we have used in this chapter. It is not difficult to see that these definitions are equivalent. □

5.4 Internet versus Wireless Formulations: An Example

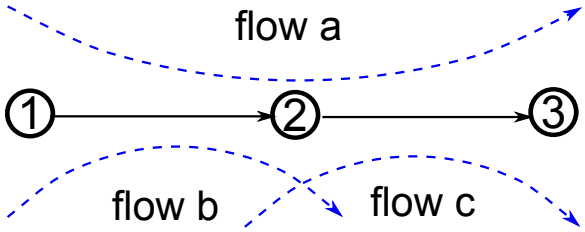

Figure 5.8: A simple line network with two links and three flows

Consider a simple 2-link, 3-flow network as shown in Figure 5.8. Assume that the two links can operate simultaneously. For example, it could be a wireline network or two wireless links operating in different frequency bands so that they do not interfere with each other. Let both links have capacity of one.

Then, in the network utility maximization formulations in this chapter, we set up the following resource allocation problem:

$$\textbf{Formulation 1:} \quad \max_{x,\mu \ge 0} \sum_{f=a,b,c} U_f(x_f)$$
$$x_a \le \mu_{12}^{(3)}$$
$$x_b \le \mu_{12}^{(2)}$$
$$x_c + \mu_{12}^{(3)} \le \mu_{23}^{(3)}$$
$$\mu_{12}^{(2)} + \mu_{12}^{(3)} \le 1$$
$$\mu_{23}^{(3)} \le 1,$$

where the x's are the flow rates and the μ's are the data rates allocated to per destination queues at each link. As we have seen, this results in congestion control, which acts on the ingress queues at the sources, along with a back-pressure algorithm as shown in Figure 5.9. In this case, the back-pressure algorithm is not used to schedule links since two links can be on simultaneously. However, it determines which queue will be served on each link at each time instant.

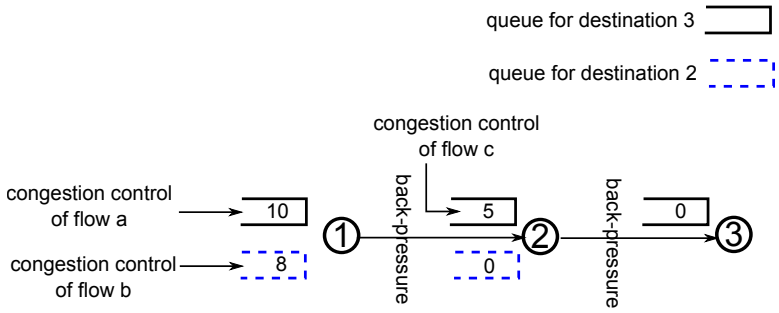

Figure 5.9: The NUM architecture of Formulation 1

An alternative formulation was presented in Chapter 1, which is:

$$\textbf{Formulation 2:} \quad \max_{x \geq 0} U_a(x_a) + U_b(x_b) + U_c(x_c)$$
$$x_a + x_b \leq 1$$
$$x_a + x_c \leq 1.$$

This results in a congestion control algorithm which simply uses the price feedback from the network. There was no back-pressure algorithm. In other words, we did not need to maintain separate queues for different destinations, and all packets can be stored in a FIFO queue as shown in Figure 5.10. We next discuss the difference between these two formulations.

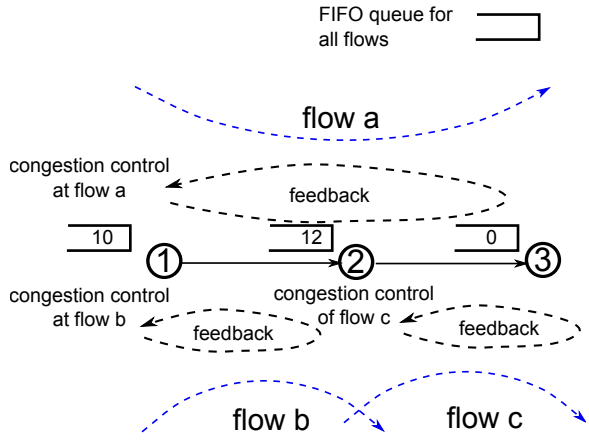

Figure 5.10: The NUM architecture of Formulation 2

In Formulation 1, we correctly model the fact that a flow's packets do not arrive at each of the links in its path instantaneously. Flow a's packets have to arrive at node 1, then at node 2 and

finally at node 3 to depart the network. So the departures from the first link are the arrivals to the second link. Hence, the constraints are

$$x_a \leq \mu_{12}^{(3)} \quad \text{and} \quad x_b + \mu_{12}^{(3)} \leq \mu_{23}^{(3)}.$$

On the other hand, in Formulation 2, we assume that packets from flow a arrive at all the links on its path simultaneously. This leads to a simpler algorithm.

While Formulation 1 is exact, we believe that Formulation 2 is appropriate for the wired Internet. The reason is that the Internet, except at access points, operate at less than full capacity. Hence, queues tend to be small or at least can be made small with a well-designed algorithm. So the delays are small and it seems reasonable to assume that packets arrive instantaneously at all links in its path.

On the other hand, in wireless networks, such an assumption may not be reasonable. Due to interference constraints (for instance, if the two links in this example can not transmit simultaneously), their queues will accumulate at one link when the other is being scheduled. So Formulation 1 may be more reasonable.

In conclusion, Formulation 1 is more reasonable in wireless networks with interference, while Formulation 2 seems to be appropriate for wireline networks and leads to simpler algorithm.