1 Utility Maximization in Networks

We now consider the resource allocation problem in a communication network such as the Internet. Suppose we have a network with a set of traffic sources $S$ and a set of links $L$. Each link $l \in L$ has a finite fixed capacity $c_l$. Each source in $S$ is associated with a route $r \subset L$ along which it transmits at some rate $x_r$. 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 $L$. 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 feel the effect of a rate increase from 0 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.

It is straightforward to write down the problem as an optimization problem of the form

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

subject to the constraints

$$\sum_{r \in r} x_r \leq c_l, \forall l \in L,$$

$$x_r \geq 0, \forall r \in S.$$  

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 $r$. Then, from a well-known property of concave functions, the optimal rates which solve the network utility maximization problem, $\{\hat{x}_r\}$, satisfy

$$\sum_r \frac{x_r - \hat{x}_r}{\hat{x}_r} \leq 0,$$

where $\{x_r\}$ is any other set of feasible rates. The left-hand side of the above expression is nothing but the inner product of the gradient vector at the optimal allocation and the vector of deviations from the optimal allocation. This inner product is non-positive for concave functions. 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 $w_r$ 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 $\{\hat{x}_r\}$ 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., $x_s > \hat{x}_s$, then there has to be another user $u$ with the property

$$x_u < \hat{x}_u \quad \text{and} \quad \hat{x}_u \leq \hat{x}_s.$$
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
\[
\min_r \hat{x}_r \geq \min_r x_r,
\]
for any other allocation \( \{x_r\} \). To see why this is true, suppose that exists an allocation such that
\[
\min_r \hat{x}_r < \min_r x_r. \tag{4}
\]
This implies that, for any \( s \) such that \( \min_r \hat{x}_r = \hat{x}_s \), the following holds: \( \hat{x}_s < x_s \). Otherwise, our assumption (4) cannot hold. However, this implies that if we switch the allocation from \( \{\hat{x}_r\} \) 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 \( \{\hat{x}_r\} \)). 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, the name minimum potential delay fairness.

All of the above notions of fairness can be captured by using utility functions of the form
\[
U_r(x_r) = \frac{x_r^{1-\alpha}}{1-\alpha}, \tag{5}
\]
for some \( \alpha_r > 0 \). Resource allocation using the above utility function is called \( \alpha \)-fair. Different values of \( \alpha \) 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 provide a heuristic argument to show that the limit \( \alpha \to \infty \) gives max-min fairness. Let \( \hat{x}_r(\alpha) \) be the \( \alpha \)-fair allocation. Then, by the property of concave functions mentioned at the beginning of this section,
\[
\sum_r \frac{x_r - \hat{x}_r}{\hat{x}_r^\alpha} \leq 0.
\]
Considering an arbitrary flow $s$, the above expression can be rewritten as

$$\sum_{r: \hat{x}_r \leq \hat{x}_s} (x_r - \hat{x}_r) \frac{\hat{x}_s}{x_r} + (x_s - \hat{x}_s) + \sum_{i: \hat{x}_i > \hat{x}_s} (x_i - \hat{x}_i) \frac{\hat{x}_s}{x_i} \leq 0.$$  

If $\alpha$ is very large, one would expect the third term in the above expression to be negligible. Thus, if $x_s > \hat{x}_s$, then the allocation for at least one user whose rate satisfies $\hat{x}_r \leq \hat{x}_s$ will decrease.

### 2 The Primal Algorithm for Distributed Utility Maximization

In our solution to the network utility maximization problem for a simple network in the previous chapter, we assumed the existence of a central authority that has complete information about all the routes and link capacities of the network. This information was used to fairly allocate rates to the different sources. Clearly, such a centralized solution does not scale well when the number of sources or the number of nodes in the network becomes large. We would like to design algorithms in which the sources themselves perform calculations to determine their fair rate based on some feedback from the network. We would also like the algorithms to be such that the sources do not need very much information about the network. How would one go about designing such algorithms?

We relax the optimization problem in several ways so as to make algorithm design tractable. The first relaxation is that instead of directly maximizing the sum of utilities constrained by link capacities, we associate a cost with overshooting the link capacity and maximize the sum of utilities minus the cost. In other words, we now try to maximize

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

where $x$ is the vector of rates of all sources and $B_l(.)$ is either a “barrier” associated with link $l$ which increases to infinity when the arrival rate on a 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 chapter; for example, choose $B_l(y)$ to be zero if $y \leq c_l$ and equal to $\infty$ if $y \geq c_l$. 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. In this case, one may wish to heavily penalize the system if the arrival rate is larger than say 90\% of the link capacity. Losing a small fraction of the link capacity may be considered to be quite reasonable if it improves the quality of service seen by the users of the network since increasing the link capacity is quite inexpensive in today’s Internet.

Another relaxation that we make is that we don’t require that the solution of the utility maximization problem be achieved instantaneously. We will allow the design of dynamic algorithms that asymptotically (in time) approach the required maximum.

Now let us try to design the algorithm that would satisfy the above requirements. Let us first consider the penalty functions $B_l(.)$ that appears in 6 above. It is reasonable to require that $B_l$ is a convex function since we would like the penalty function to increase rapidly as we approach or exceed the capacity. Further, assume that $B_l$ is continuously differentiable. Then, we can equivalently require that

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

(7)
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. If \( f_l \) is differentiable, it is easy to check the convexity of \( B_l \) since the second derivative of \( B_l \) would be positive since \( f_l \) is an increasing function.

Since \( U_r \) is strictly concave and \( B_l \) is convex, \( V(x) \) strictly concave. Further, we assume that \( U_r \) and \( f_l \) are such that the maximization of (6) results in a solution with \( x_r \geq 0 \) \( \forall r \in S \). Now, the condition that must be satisfied by the maximizer of (6) is obtained by differentiation and is given by

\[
U'_r(x_r) - \sum_{l : l \in r} f_l \left( \sum_{s : l \in s} x_s \right) = 0, \quad r \in S. \tag{8}
\]

We now require an algorithm that would drive \( x \) towards the solution of (8). A natural candidate for such an algorithm is the 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 \( \delta \to 0 \)

\[
\dot{x} = k(t) \frac{dg(x)}{dx}.
\tag{9}
\]

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

\[
\dot{x}_r = k_r(x_r) \left( U'_r(x_r) - \sum_{l : l \in r} f_l \left( \sum_{s : l \in s} x_s \right) \right), \tag{10}
\]

where \( k_r(\cdot) \) is non-negative, increasing and continuous. We have obtained the above by differentiating (6) 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 (9). Clearly, the stationary point of the above algorithm satisfies (8) and hence maximizes (6). 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 \( q_r = \sum_{l : l \in r} f_l \left( \sum_{s : l \in s} x_s \right) \) 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 stationary 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 sub-section.

The question of information required is easily answered by studying (10). 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 \( \text{ack} \) 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

\[
y_l = \sum_{s: l \in s} x_s, \tag{11}
\]

which is the load on link \( l \). Using the elements of the routing matrix and recalling the notation of Section 1, \( y_l \) can also be written as

\[
y_l = \sum_{s: l \in s} R_{ls} x_s.
\]

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

\[
y = Rx \tag{12}
\]

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

\[
p_l(t) = f_l \left( \sum_{s: l \in s} x_s \right) = f_l(y_l(t)). \tag{13}
\]

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

\[
q_r = \sum_{l: l \in r} p_l(t). \tag{14}
\]

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

\[
q = R^Tp \tag{15}
\]

The above is more than just an intellectual exercise, since the expressions (12) and (15) provide linear relationships between the control at the sources and the control at the links that will help us later in analyzing the system further. The relationships derived above can be made clear using the block diagram in Figure 1.

To answer the question of whether or not our controller actually achieves the desired allocation, we have to study the properties of the controller dynamics. For this purpose, we use the concept of a Lyapunov function, which is widely used in control theory.

Recall that \( V(x) \) is a strictly concave function. Let \( \hat{x} \) be its unique maximum. Then, \( V(\hat{x}) - V(x) \) is non-negative and is equal to zero only at \( x = \hat{x} \). Thus, \( V(\hat{x}) - V(x) \) is a natural candidate Lyapunov function for the system (10). We use this Lyapunov function in the following theorem.
Figure 1: A block diagram view of the congestion control algorithm. The controller at the source uses congestion feedback from the link to perform its action.

**Theorem 1** Consider a network in which all sources follow the primal control algorithm (10). Assume that the functions $U_r(\cdot)$, $k_r(\cdot)$ and $f_l(\cdot)$ are such that $W(x) = V(\hat{x}) - V(x)$ is such that $W(x) \to \infty$ as $||x|| \to \infty$, $\hat{x}_i > 0$ for all $i$, and $V(x)$ is as defined in (6). Then, the controller in (10) is globally asymptotically stable and the equilibrium value maximizes (6).

**Proof** Differentiating $W(\cdot)$, we get

$$\dot{W} = -\sum_{r \in S} \frac{\partial V}{\partial x_r} \dot{x}_r = -\sum_{r \in S} k_r(x_r) (U'_r(x_r) - q_r)^2 < 0, \forall x \neq \hat{x},$$

and $\dot{W} = \forall x = \hat{x}$. Thus, all the conditions of the Lyapunov theorem are satisfied and we have proved that the system state converges to $\hat{x}$.

In the proof of the above theorem, we have assumed that 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

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

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

### 3 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 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

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

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 $\sum_{l \in r} p_l$. The assumption here is that the congestion control occurs at a slower time-scale than the packet dynamics in the network so that $p_l$ remains roughly a constant over many packets.

Price functions were assumed to be functions of the arrival rate at a link. However, in practice, it is quite common to indicate congestion based on the queue length at a link. For example, consider the following congestion notification mechanism. Whenever the queue length at a node exceeds a threshold $B$ at the time of a packet arrival, then the packet is marked. For the moment assume that there is only one link in the network. Further, again suppose that the congestion controller acts at a slower time compared to the queue length dynamics. For example, the congestion controller may react to the fraction of packets marked over a time interval. In this case, one can roughly view the price function for the link as the fraction of marked packets. To compute the fraction of marked packets, one needs a model for the packet arrival process at the link which could, in general, be a stochastic process. For simplicity, assume that the arrival process is a Poisson process of rate $y_l$ which remains constant compared to the time-scale of the congestion control process. Further, suppose that packet sizes are exponentially distributed. Then, the fraction of marked packets is simply the probability that the number of packets in an $M/M/1$ queue exceeds $B$, which is given by $p_l = f_l(y_l) = (y_l/c_l)B$ if $y_l < c_l$ and is equal to 1 otherwise (to be shown in a later lecture). If there is only one bit in the packet header to indicate congestion, then a packet is marked if it is marked on any one of the links on its route. Thus, the packet marking probability on a route $r$ is given by $1 - \prod_{l \in r} (1 - p_l)$. If the $p_l$ are small, then one can approximate the above by $\sum_{l \in r} p_l$, which is consistent with the primal formulation of the problem.

Instead of simply marking packets when the queue length exceeds a threshold, other price functions have been suggested in the literature as well. A well-known such mechanism is RED (random early detection) scheme. Let $b_l$ be the number of packets in the buffer at link $l$. Then, the RED packet marking scheme is given by

$$g(b_l) = \begin{cases} 
0, & \text{if } b_l \leq B_{\text{min}} \\
K(b_l - B_{\text{min}}), & \text{if } B_{\text{min}} < b_{\text{av}} \leq B_{\text{max}} \\
1, & \text{if } b_{\text{av}} > B_{\text{max}}
\end{cases}$$

(17)

where $K$ is constant, and $B_{\text{min}}$ and $B_{\text{max}}$ are fixed thresholds. This price function also lies strictly in $[0, 1]$ and is illustrated in Figure 2. Again, if the packet-level dynamics in the queue are faster than the congestion-control dynamics, then one can convert the above marking scheme into a price function that depends on the link rate. Under the $M/M/1$ assumption as before, the probability that there are $n$ packets in the buffer is given by

$$r_n = \rho_l (1 - \rho_l)^n,$$

where $\rho_l = y_l/c_l$. Then, the equivalent price function for the RED scheme is given by

$$f_l(y_l) = \sum_{n=0}^{\infty} r_n g(n)$$
if $\rho_l < 1$ and is equal to 1 if $\rho_l \geq 1$. In general, given any queue-based marking function $g_l(b_l)$, one can obtain an equivalent rate-based price function by computing $E(g_l(b_l))$ where $b_l$ is the steady-state queue length of a queueing model with arrival rate $y_l$ and service rate $c_l$. In the above calculations, we have assumed that the queueing model is an $M/M/1$ model, but one can use other, more complicated models as well. A queue-based marking scheme is also called an active queue management or AQM scheme.

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. Assuming Poisson arrivals and exponentially distributed file sizes, the probability that a packet is dropped when the buffer size is $B$ is given by

$$1 - \frac{1}{1 - \rho} \frac{\rho^B}{B!}$$

where $\rho = \sum_{r \in \mathcal{R}} x_r$. As the number of users of the Internet increases, one might conceivably increase the buffer sizes as well in proportion, thus maintaining a constant maximum delay at each link. In this case, $B \to \infty$ would be a reasonable approximation. We then have

$$\lim_{B \to \infty} \frac{1 - \rho}{1 - \rho^{B+1}} \rho^B = \begin{cases} 0, & \text{if } \rho < 1, \\ 1 - \frac{1}{\rho}, & \text{if } \rho \geq 1. \end{cases}$$

Thus, an approximation for the drop probability is $(1 - \frac{1}{\rho})^+$, which is non-zero only if $\sum_{r \in \mathcal{R}} x_r$ 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 link 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. Thus, the optimization is valid only if the drop probability is small. 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.