IEICE TRANS. COMMUN., VOL.E96–B, NO.11 NOVEMBER 2013

INVITED PAPER *Special Section on Progress in Information Network Science*

# Congestion Control, Routing and Scheduling in Communication Networks: A Tutorial

**Jean WALRAND**[†a)] ***and*** **Abhay K. PAREKH**[†], ***Nonmembers***

**SUMMARY** In communication networks, congestion control, routing, and multiple access schemes for scheduling transmissions are typically regulated by distributed algorithms. Engineers designed these algorithms using clever heuristics that they refined in the light of simulation results and experiments. Over the last two decades, a deeper understanding of these algorithms emerged through the work of researchers. This understanding has a real potential for improving the design of protocols for data centers, cloud computing, and even wireless networks. Since protocols tend to be standardized by engineers, it is important that they become familiar with the insights that emerged in research. We hope that this paper might appeal to practitioners and make the research results intuitive and useful. The methods that the paper describes may be useful for many other resource allocation problems such as in call centers, manufacturing lines, hospitals and the service industry.

***key words:*** *distributed algorithm, congestion control, routing, scheduling, multiple access, utility maximization, backpressure, entropy relaxation*

## 1. Introduction

In the early 1970s, Robert Metcalfe [13] designed a randomized multiple access scheme for devices attached to a common cable to regulate their sending of packets. This scheme is a distributed scheduling mechanism that decides when devices can transmit. The basic idea is that a device should transmit when the cable is idle and repeat its attempt after a random time if the transmission fails, presumably because it collided with another transmission. Because the devices double the randomization interval after each collision, the scheme is called binary exponential back-off. Around the same time, Vint Cerf and Bob Kahn developed the TCP/IP protocols that now form the basis of the Internet [4]. In the mid 1980s, it became apparent that TCP was not regulating the flows of packets appropriately. Van Jacobson [6] designed an improved algorithm, based on the additive-increase, multiplicative decrease (AIMD) scheme studied by Dah Ming Chiu and Raj Jain [5]. Essentially, if a host that is sending packets senses that the network is congested, it slows down; otherwise, it keeps speeding up.

These two schemes, binary exponential back-off and AIMD, are examples of distributed algorithms. Although their basic schemes are intuitive and sensible, it is hard to analyze their delay, throughput, and fairness (see e.g., [2]). Consequently, it is difficult to know whether a variation on the basic algorithms would perform substantially better. In fact, a considerable amount of work has been devoted to tweaking these two schemes and much work is still going on along those lines. On a pragmatic basis, one may argue that the binary exponential back-off scheme that is now at the heart of Wi-Fi works well enough and that TCP is too widely used to even think about modifying it. However, the increasing use of interactive applications in the Internet and the importance of data centers may justify another look at these mechanisms. It is also quite plausible that new applications will emerge that benefit from tighter control on delays and throughput. With these objectives in mind, we propose a guided tour through the lessons that researchers have learned over the last twenty years on how to design distributed algorithms for networks. Although the methods that the paper describe apply to a wide class of resource allocation problems, we focus on congestion control, routing and resource scheduling. A more detailed presentation of these topics can be found in [18].

Manuscript received March 18, 2013.
Manuscript revised June 15, 2013.

[†]The authors are with the Department of EECS, University of California, Berkeley, USA.
a) E-mail: wlr@eecs.berkeley.edu

DOI: 10.1587/transcom.E96.B.2714

## 2. Congestion Control

Consider three connections that share communication links with transmission rates $C_1$ and $C_2$, as shown in Fig. 1. Each source implements a congestion control algorithm that uses only information locally available to regulate the rate at which it sends packets to the destination. The goal is to share the links efficiently (fully) and fairly.

### 2.1 Efficiency and Fairness

What is the meaning of fairness and why should the network be fair? The network provides a connection service to different applications or different users. Intuitively, one wants all the applications to work well. Otherwise, users may switch to another network.

For instance, one could design the network to maxi-

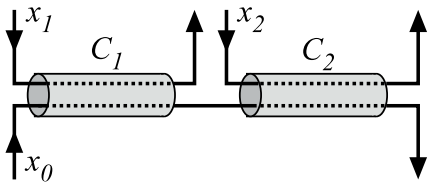


**Fig. 1** Three connections share two links. Here, for $i = 0, 1, 2$, $x_i$ indicates the average transmission rate of connection $i$, in bits per second. Also, for $j = 1, 2$, $C_j$ indicates the transmission rate of link $j$.

mize the sum $x_0 + x_1 + x_2$ of the rates of the three connections. For concreteness, assume that $C_1 = 8$ and $C_2 = 4$ in the network of Fig. 1. To maximize the sum of the rates $x_0 + x_1 + x_2$, one should select

$$x_0 = 0, x_1 = C_1 = 8, x_2 = C_2 = 4.$$

However, this *max-sum allocation* is unfair to the user of connection 0, and this is not acceptable.

An alternative design objective might be to maximize the minimum $\min\{x_0, x_1, x_2\}$ of the rates. The solution is then

$$x_0 = 2, x_1 \in [2, 6], x_2 = 2.$$

Indeed, increasing $x_0$ would decrease $x_2$ because $x_0 + x_2 \leq 4$, and would thus reduce the minimum rate. This *max-min allocation* is somewhat unfair to connection 2 because it uses less resources than connection 0, so that it may make sense to have an allocation where $x_0 < x_2$.

Another possible allocation is one that maximizes the product $x_0 x_1 x_2$ of the rates. One can verify that it corresponds to

$$x_0 = 1.7, x_1 = 6.3, x_2 = 2.3.$$

This allocation, called *proportionally fair* or also the *Nash bargaining equilibrium*, has the property that deviating from it would corresponds to a negative sum

$$\sum_{i=0}^{2} \frac{\Delta x_i}{x_i}$$

of fractional increases in rates. Thus, one might be able to increase the rate of one user by 2%, but at the cost of decreasing the rates of the other two users by 1.3% and 0.8%, say. One may argue that it is not fair for the other users to "pay" more (in fractional change) than the fractional gain of one user. Note that maximizing the product of the rates is the same as maximizing the sum of their logarithms, i.e., to maximize

$$\sum_{i=0}^{2} \log(x_i).$$

Yet another possibility is to maximize

$$\sum_{i=0}^{2} \frac{x_i^{1-\alpha}}{1-\alpha}$$

where $\alpha > 0$ and $\alpha \neq 1$. For $\alpha = 1$, one maximizes the sum of the logarithms. The resulting solution is called an *$\alpha$-fair allocation* (see [14]). As $\alpha$ increases, the function $x^{1-\alpha}/(1-\alpha)$ gets steeper for small $x$ and flatter for large $x$. Accordingly, the allocation that maximizes the sum above goes from *max-sum* to *max-min* as $\alpha$ increases from 0 to $\infty$, which is a way to select a trade-off between efficiency and fairness. Figure 2 shows how $x_0$ changes as a function of $\alpha$.

Thus, a systematic way of selecting the rates is to find

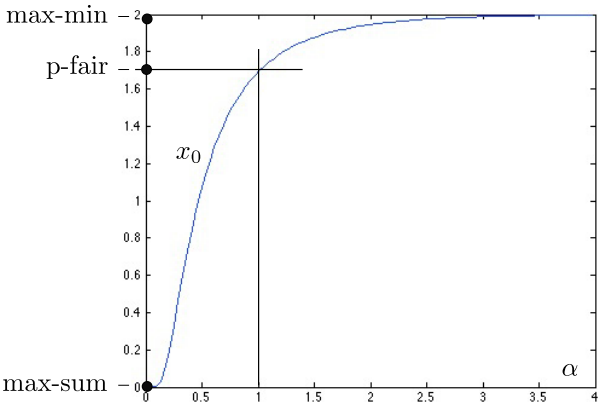


**Fig. 2** $\alpha$-fair allocation. The figure shows $x_0$ on the $Y$-axis as a function of $\alpha$ on the $X$-axis. That value goes from 0 when $\alpha = 0$, which corresponds to max-sum, to 1.7 when $\alpha = 1$, which is proportionally fair, to 2 when $\alpha \to \infty$, which is max-min fair.

the values that maximize the sum of the utilities of the connections, i.e., to solve

$$\text{Maximize } \sum_i U_i(x_i)$$

subject to capacity constraints.

This is the formulation that Frank Kelly introduced in the late 1990s [9]–[10]. The functions $U_i(x_i)$ are concave increasing functions such as $\log(x_i)$ or $x_i^{1-\alpha}/(1-\alpha)$. The concavity captures the diminishing value of additional rate when the rate increases. An economic interpretation is that $U_i(x_i)$ is how much user $i$ would be willing to pay for the rate $x_i$.

## 2.2 AIMD

The additive increase, multiplicative decrease (AIMD) scheme underlies the transmission control protocol TCP of the Internet (see [5], [6]). The rules of this scheme are as follows:

- as long as a source gets acknowledgments, it increases its transmission rate;
- when it misses acknowledgments, a source divides its rate by a factor 2.

More precisely, the source increases its *congestion window size* by one unit per round-trip time. One unit is a fixed number of bytes and the round-trip time is the time to get an acknowledgment after sending a packet.

Figure 3 sketches how the average rates of the three connections change over time when the sources use AIMD. The figure assumes that $C_1 = 8, C_2 = 4$ and that the round-trip time is twice as long for connection 0 as it is for connections 1 and 2. The figure shows that the long term average rates of the three connections are

$$x_0 = 0.47, x_1 = 5.65, x_2 = 2.77.$$

Thus, the sum of the average rates for the two links are 6.12

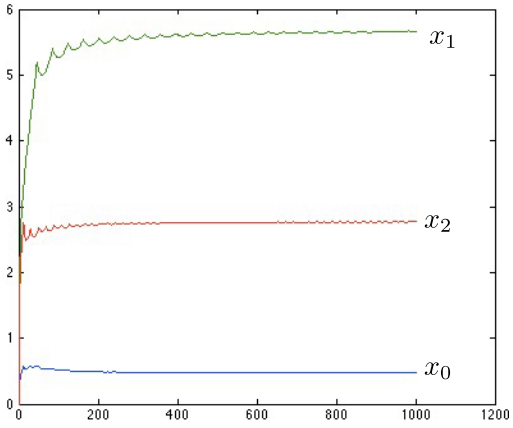


**Fig. 3** Evolution of rates using AIMD. The figure shows the rates $x_0, x_1, x_2$ of the three connections, on the $Y$-axis, as a function of time, on the $X$-axis, as the algorithm adjusts these rates. The limiting rates show that AIMD is severely biased against connections with a larger round-trip time.

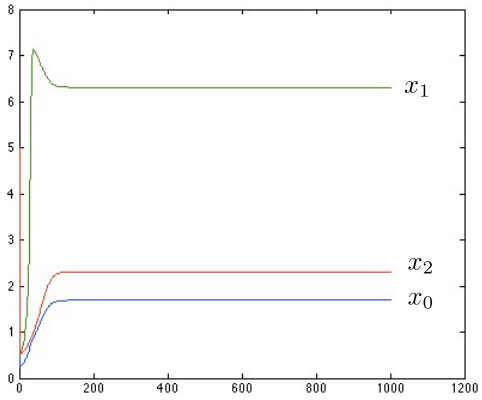


**Fig. 4** Evolution of rates using the proportional fair algorithm. The figure shows the rates $x_0, x_1, x_2$ of the three connections, on the $Y$-axis, as a function of time, on the $X$-axis, as the algorithm adjusts these rates. The limiting rates are proportionally fair.

and 3.24, respectively, which is about 80% of the capacities of the two links. The loss in throughput in our model comes from the fact that the buffers are too small and it disappears if one uses larger buffers, which is done in practice.

We see that connection 0 has a very small throughput, partly because it uses two links and partly because its long round-trip time results in slower increases in the rate of its connection, which allows the other connections to take better advantage of available capacity.

Thus, AIMD is a clever ad hoc scheme that stabilizes the rates of the connections so that no router gets saturated, and this was the main goal of its inventor Van Jacobson [6]. However, the scheme results in connection rates that depend strongly on the round-trip times and on the number of links they go through.

### 2.3 Utility Maximization

Consider Kelly's utility maximization problem with $U_i(x) = \log(x)$. The solution, as we know is the proportional fair allocation. We are looking for a distributed algorithm that finds that allocation. That is, we consider

$$\begin{aligned} &\text{Maximize } g(\mathbf{x}) := \log(x_0) + \log(x_1) + \log(x_2) \\ &\text{subject to } x_0 + x_1 \le C_1 \text{ and } x_0 + x_2 \le C_2. \end{aligned} \tag{1}$$

Here, $\mathbf{x} := (x_0, x_1, x_2)$.

Instead of solving this problem in a centralized way, the network can use a distributed algorithm based on a simple feedback mechanism. Each link $j = 1, 2$ charges a price $\beta q_j$ per unit rate where $q_j$ is the backlog in buffer $j$ and $\beta$ is a small positive constant. Each connection chooses the rate that maximizes the utility of that rate minus its cost. As a result, the connections slow down when the backlogs increase, which was also the goal of AIMD. Note that the algorithm is distributed in the sense that the users do not know about each other and react only to the prices of the links they use. Also, the links calculate their prices without knowing about their users. Moreover, the algorithm does not require any information about the topology of the network, the rates of the links, or the paths of the connections.

Thus, for $i = 0, 1, 2$,

$$x_i \text{ maximizes } \log(x_i) - \gamma_i x_i.$$

In this expression, $\gamma_i$ is the price per unit rate of connection $i$, which is proportional to the sum of the backlogs of the links it uses. Thus,

$$\gamma_0 = \beta q_1 + \beta q_2, \gamma_1 = \beta q_1, \text{ and } \gamma_2 = \beta q_2.$$

In particular, note that connection 0 adjusts its rate $x_0$ based on the sum of the backlogs in the links that it goes through. One can argue that TCP does something similar. Recall that when using TCP, a source slows down when it notices that the path is congested, either because of missing acknowledgments or because of marked acknowledgments. Thus, the source reduces the rate of a connection in response to the total congestion along the path. However, the AIMD implementation of this rate adjustment is a bit too crude to maximize the sum of the utilities, as we noticed in our simulation. Obviously, modifying TCP in the Internet is not feasible because the suitable modifications are not backward compatible.

Figure 4 shows the evolution in time of the rates of the three connections when this algorithm is used. This algorithm converges to the proportionally fair allocation $x_0 = 1.7, x_1 = 6.3, x_2 = 2.3$.

There are two approaches to justify this algorithm. The first one is based on a dual algorithm for convex optimization. The second uses is based on a Lyapunov function. We explain these two approaches in the next sections.

### 2.4 Dual Algorithm

This algorithm is based on the fact that the solution $\mathbf{x}^*$ of problem (1) is such that

$$L(\mathbf{x}, \lambda^*) \le L(\mathbf{x}^*, \lambda^*) \le L(\mathbf{x}^*, \lambda), \forall \mathbf{x}, \lambda$$

where

$$L(\mathbf{x}, \lambda) = g(\mathbf{x}) - \lambda_1(x_0 + x_1 - C_1) - \lambda_2(x_0 + x_2 - C_2). \quad (2)$$

The interpretation of $L(\mathbf{x}, \lambda)$ is that the capacity constraint $x_0 + x_1 \le C_1$ is replaced by a cost $\lambda_1(x_0 + x_1 - C_1)$ for exceeding the constraint, and similarly for the capacity constraint of link 2. The constants $\lambda_1$ and $\lambda_2$ are called the *Lagrange multipliers* or *shadow prices* of the constraints. The intuition is that if one chooses these shadow prices suitably, then the values of the rates $x_i$ that maximize $L(\mathbf{x}, \lambda)$ are such that the constraints are satisfied exactly. See [3] for a discussion of convex optimization algorithms.

Thus, to find $\mathbf{x}^*$, one maximizes $L(\mathbf{x}, \lambda)$ over $\mathbf{x}$ and to find $\lambda^*$, one minimizes $L(\mathbf{x}, \lambda)$ over $\lambda$. The first step leads to

$$x_i \text{ maximizes } \log(x_i) - \gamma_i x_i$$

where $\gamma_0 = \lambda_1 + \lambda_2, \gamma_1 = \lambda_1$, and $\gamma_2 = \lambda_2$. The second step, minimizing over $\lambda$ is done by a gradient projection algorithm where one updates $\lambda_i$ in the direction opposite to the derivative of $L$ with respect to $\lambda_i$. Thus, if $\lambda_i(n)$ is the value of $\lambda_i$ at step $n$ of the gradient projection algorithm, one has

$$\lambda_i(n+1) = \max\left\{0, \lambda_i(n) - \alpha_n \frac{\partial}{\partial \lambda_i} L(\mathbf{x}, \lambda(n))\right\}$$

where $\alpha_n > 0$ determines the size of the $n$-th step of he algorithm. For instance,

$$\lambda_1(n+1) = \max\{0, \lambda_1(n) + \alpha_n[x_0 + x_1 - C_1]\}.$$

Note that this update changes $\lambda_1(n)$ according to the excess of the arrival rate over the service rate at buffer 1. Hence, one expects $\lambda_i$ to increase when the backlog at link $i$ increases. See [10] and [12] for a detailed analysis of these algorithms. One can show that if $\alpha_n = 1/n$, then this algorithm converges to the rates $\mathbf{x}^*$ that solve the optimization problem.

### 2.5 Lyapunov Method

Instead of the dual algorithm, an alternative analysis is based on the drift of a *Lyapunov function* for the network. A Lyapunov function is a nonnegative function of the state of the system that tends to decrease when the state is outside a finite set. This property implies that the state returns periodically to that finite set, so that the system is stable. The method provides a guideline for designing the control scheme for the system by making the drift of the Lyapunov function as negative as possible, to enforce the stability of the system. As we explain, the Lyapunov method yields prices for the links that are proportional to the backlogs and it results, for many stochastic systems, in a utility that is close to the optimal value. When using the dual algorithm, the prices have to be calculated recursively, but its performance converges to the optimal one. These two methods provide complementary ways to design dynamic resource allocation schemes.

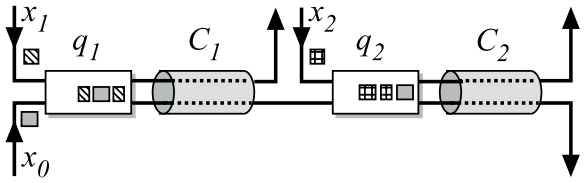


**Fig. 5** Each link is equipped with a FCFS queue. As before, $x_i$ is the rate of connection $i$ ($i = 0, 1, 2$) and $q_j$ is the backlog in the buffer of link $j$ ($j = 1, 2$).

To explain this method, we redraw the network of Fig. 1 as Fig. 5 where we show the buffers (queues) that store packets before they are transmitted on the links. Each link $j$ (for $j = 1, 2$) is equipped with a queue and we indicate by $q_j$ the backlog (in bits) in the queue of link $j$. Note that the packets of connections 0 and 1 are stored in the same queue and they are served on a first-come, first served (FCFS) basis by the transmitter that sends them one by one through link 1, and similarly for link 2.

The Lyapunov function is $V(\mathbf{q}) := (1/2)(q_1^2 + q_2^2)$, where $\mathbf{q} := (q_1, q_2)$. Thus, $V(\mathbf{q})$ is large when the backlogs in the queues are large. The control that one derives attempts to reduce $V(\mathbf{q})$, to make the queues small, and yet achieve a large utility $g(\mathbf{x})$. To achieve a suitable trade-off between these conflicting goals, one considers the following combination of the utility of the rates $x_i$ and the drift of the Lyapunov function:

$$g(\mathbf{x}(t)) - \alpha \frac{d}{dt} V(\mathbf{q}(t)).$$

In this expression, $\mathbf{x}(t)$ is the vector of rates of the connections at time $t$ and $\mathbf{q}(t)$ is the vector of backlogs at the two queues at time $t$, for $t \ge 0$. The parameter $\alpha$ selects the trade-off between congestion and utility: a large $\alpha$ favors reducing congestion whereas a small $\alpha$ gives priority to a large utility.

We consider a continuous-time model, for simplicity of exposition, even though the packets have a discrete size. Thus, we will choose $\mathbf{x}(t)$ to maximize the expression above.

Now,

$$\frac{d}{dt} V(\mathbf{q}(t)) = \frac{d}{dt} \frac{1}{2} \sum_{j=1}^{2} q_j^2(t)$$
$$= \sum_{j=1}^{2} q_j(t) \frac{d}{dt} q_j(t) = \sum_{j=1}^{2} q_j(t)[x_0 + x_j - C_j].$$

Thus, one chooses the value of $\mathbf{x}$ that maximizes

$$g(\mathbf{x}) - \alpha \sum_{j=1}^{2} q_j(t)[x_0 + x_j - C_j]. \quad (3)$$

Comparing with (2), we see that this approach corresponds to using the backlogs as "prices" for the links.

To estimate the performance of this control scheme, let $x^*$ be the solution of (1). We have

$$g(\mathbf{x}(t)) - \alpha \frac{d}{dt} V(\mathbf{q}(t))$$

$$= g(\mathbf{x}) - \alpha \sum_{j=1}^{2} q_j(t)[x_0 + x_j - C_j]$$

$$\geq g(\mathbf{x}^*) - \alpha \sum_{j=1}^{2} q_j(t)[x_0^* + x_j^* - C_j] \geq g(\mathbf{x}^*)$$

where the first inequality comes from the fact that $x(t)$ maximizes (3) and the second from the fact that $x_0^* + x_j^* \leq C_j$.

Integrating this inequality over $[0, T]$ and dividing by $T$, we conclude that

$$\frac{1}{T}\int_0^T g(\mathbf{x}(t))dt - \alpha\frac{1}{T}[V(\mathbf{q}(T)) - V(\mathbf{q}(0))] \geq g(\mathbf{x}^*).$$

Letting $T \to \infty$, we find

$$\liminf_{T\to\infty} \frac{1}{T}\int_0^T g(\mathbf{x}(t))dt \geq g(\mathbf{x}^*)$$

so that this control achieves the maximum performance.

In practice, the control is not in continuous time. A similar analysis in discrete time shows that the control achieves a performance at least equal to $g(x^*) - \alpha B$, where $B$ is a constant that depends on the maximum link rates and the time step of the control.

The idea of minimizing the drift of $V(q)$ is due to Tassiulas and Ephremides [17] where the goal was to stabilize the network. Combining this method to achieve the maximum utility by trading off drift and utility is due to Modiano, Neely and their collaborators [15]. See [16] for a detailed exposition.

### 2.6 Backpressure

In the formulation of the congestion control problem that we considered so far, each link is equipped with a single buffer and it transmits the packets in their order of arrival. An alternative design is one where each connection goes though a separate buffer and the link decides, at any given time, which buffer it serves. This increased complexity adds flexibility to the algorithms since links can now favor one connection over another, thus enabling priorities or other forms of scheduling inside the routers.

Although this design is not practical for the general Internet, it may be feasible for special networks in data centers where the symmetry of the network can be exploited to reduce the number of queues that are needed. Figure 6 shows the network of Fig. 1 modified so that each link has one queue per connection. The figure shows the backlogs $q_i$ in the corresponding four buffers. In this network, link 1 serves connection 0 at rate $a_{10}$ and connection 1 at rate $a_{11}$, and similarly for link 2. We will see that the source of each connection adjusts its rate based on the backlog of the *ingress* buffer of the connection, in contrast with TCP that adjusts it based on the total backlog along the path. Thus, the rate $x_0$ of connection 0 is based on $q_0$ instead of $q_0 + q_3$. Also, each link chooses which connection to serve at any given time, based on its backlogs and that of the next downstream buffer, hence the name *backpressure*. Specifically, we will see that the decision at link 1 to serve connection 0 or connection 1 is based on a comparison of $q_1$ and of $q_0 - q_3$.

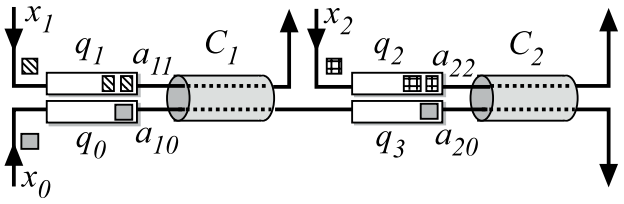


**Fig. 6** Per-flow queueing. Each link has one separate queue for each connection. Here, $q_0$ is the backlog in the queue of connection 0 at link 1, and similarly for $q_1, q_2$ and $q_3$. Also, link $j$ serves connection $i$ at rate $a_{ji}$.

As before, we consider the problem of maximizing the utility minus a multiple of the drift of $V(\mathbf{q})$ where $V(\mathbf{q})$ is half the sum of the squares of the buffer backlogs. That is, one considers the following optimization problem:

$$\text{Maximize } \sum_{i=0}^{2} U_i(x_i) - \beta\frac{d}{dt}\left[\frac{1}{2}\sum_{j=0}^{3} q_j^2(t)\right]$$
$$\text{subject to } x_0 \leq a_{10}, x_1 \leq a_{11}, a_{10} \leq a_{20}, x_2 \leq a_{22}$$
$$\text{and } a_{10} + a_{11} \leq C_1, a_{20} + a_{22} \leq C_2.$$

In this problem, the service rates of the individual buffers are control variables. For instance, link 1 may decide whether to serve packets from connection 0 or connection 1, and similarly for link 2.

We find that

$$\frac{d}{dt}q_0^2(t) = q_0(t)[x_0 - a_{10}],$$

and similarly for the other queues. Thus, one ends up with the problem of maximizing the following expression:

$$\sum_{i=0}^{2} U_i(x_i) - \beta q_0(x_0 - a_{10})$$
$$-\beta q_1(x_1 - a_{11}) - \beta q_2(x_2 - a_{22}) - \beta q_3(a_{10} - a_{20}).$$

The first step is to find the values of $\mathbf{x} = (x_0, x_1, x_2)$ and $\mathbf{a} = (a_{10}, a_{11}, a_{20}, a_{22})$ that maximize this expression subject to the two inequality constraints $a_{10} + a_{11} \leq C_1$ and $a_{20} + a_{22} \leq C_2$. Maximizing it over $\mathbf{x}$, we see that, for $i = 0, 1, 2$,

$$x_i \text{ maximizes } U_i(x_i) - \beta q_i x_i.$$

Also, maximizing the expression over $\mathbf{a}$ subject to $a_{10} + a_{11} \leq C_1$ and $a_{20} + a_{22} \leq C_2$, we see that $a_{10}$ and $a_{11}$ must maximize

$$a_{10}(\beta q_0 - \beta q_3) + a_{11}\beta q_1$$

subject to $a_{10} + a_{11} \leq C_1$. Thus,

$$(a_{10}, a_{11}) = \begin{cases} (C_1, 0), & \text{if } q_1 \geq q_0 - q_3, \\ (0, C_1), & \text{if } q_1 < q_0 - q_3. \end{cases}$$

Similarly, one finds that

$$(a_{20}, a_{22}) = \begin{cases} (C_2, 0), & \text{if } q_3 \geq q_2 \\ (0, C_2), & \text{if } q_3 < q_2. \end{cases}$$

Thus, link 1 serves buffer 1 if $q_1 > q_0 - q_3$ and serves buffer 0 otherwise. One defines the *backpressure* at one buffer as the difference between the backlog of that buffer and the backlog in the buffer to which it sends packets. If the packets leave the network from a buffer, then the backpressure of that buffer is just its backlog. Thus, link 1 serves its buffer with the maximum backpressure. Similarly, link 2 serves its buffer with the maximum backpressure. This is called a *backpressure congestion control*.

Note that each connection adjusts its rate based only on the backlog of its ingress buffer. Thus $x_0$ is adjusted based on $q_0$. Recall that in the previous formulation, $x_0$ is adjusted based on the sum of the backlogs of the links it goes through.

## 3. Routing

So far, we have considered networks where the paths of the flows were fixed. The basic routing protocols of the Internet (e.g., OSPF, RIP) select the shortest path from a source to a destination. The length of a path is the sum of the length assigned to the links along the path. The length of a link is a number that decreases with the link rate. Thus, a 10-Gbps link is assigned a length that is shorter than a 100-Mbps link. The length is fixed; it does not depend on the backlog at the buffer of the link nor on the traffic on that link or the physical length of the link. In particular, the Internet does not perform any adaptive routing based on congestion or on the requirements of the applications. Some protocols (traffic engineering in MPLS, BGP) enable the network providers to tune the routing on the basis of their knowledge of the traffic and of some other policies such as transit or peering agreements or security considerations. In all cases, the routing decisions are quasi-static and do not change in real-time as the congestion changes. Recent trends, such as software-defined networks, facilitate the setting of routing policies, so that they could be adjusted more rapidly, but still in a quasi-static manner.

In this section, we examine how the systematic formulation of a network utility maximization problem can guide the design of routing policies. The result will be dynamic routing schemes that may be useful in specialized situations, such as task allocations in data centers.

Figure 7 shows a simple network with three flows of packets. Flow 0 has two possible paths: through link 1 or link 2. That is, the transmitter at link 0 can label the packets are being destined to link 1 or to link 2. For simplicity, we do not show the switch that guides these packets to the appropriate link. The figure shows the backlogs and the service rates of the five buffers. In particular, note that buffer 0 sends packets at rate $a_1$ to buffer 1 and at rate $a_2$ to buffer 2. That is, buffer 0 performs some dynamic routing decisions.

The problem is to choose the connection rates, the routing decisions, and the service rates at the links to maximize the total utility of the users. We will see that, as in the backpressure discussion, each connection adjusts its rate based on its ingress buffer. Also, each node chooses which buffer to serve based on the backpressures. In particular, node 0 ends up routing flow to the shortest downstream queue ($q_3$ or $q_4$) or it stops sending packets.

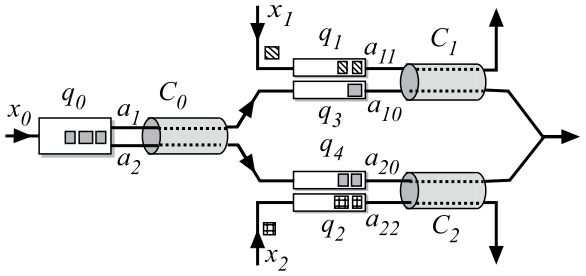


**Fig. 7** Network with routing. As before, the links use per-flow queues. Link 0 decides the rates $a_1$ and $a_2$ at which it sends packets to links 1 and 2, thus making routing decisions.

One formulates the problem as follows:

$$\text{Maximize } \sum_{i=0}^{2} U_i(x_i) - \beta \frac{d}{dt}\left[\frac{1}{2}\sum_{j=0}^{4} q_j^2(t)\right]$$
$$\text{subject to } x_0 \leq a_1 + a_2, x_1 \leq a_{11}, x_2 \leq a_{22}$$
$$\text{and } a_1 \leq a_{10}, a_2 \leq a_{20},$$
$$a_1 + a_2 \leq C_0, a_{11} + a_{10} \leq C_1,$$
$$a_{22} + a_{20} \leq C_2.$$

Note that

$$\frac{1}{2}\frac{d}{dt}\sum_{j=0}^{4} q_j^2(t) = q_0(x_0 - a_1 - a_2) + q_1(x_1 - a_{11}) + q_2(x_2 - a_{22}) + q_3(a_1 - a_{10}) + q_4(a_2 - a_{20}).$$

As in our study of backpressure, we find that maximizing over $x_i$ gives

$$x_i \text{ maximizes } U_i(x_i) - \beta q_i x_i.$$

That is, connection $i$ adjusts its rate based on the backlog of its ingress buffer, not on the sum of the backlogs of the links it goes through.

To maximize the expression over $(a_1, a_2)$ subject to $a_1 + a_2 \leq C_0$, one maximizes

$$a_1(q_0 - q_3) + a_2(q_0 - q_4)$$

subject to that constraint. That is, one has to solve the following problem:

$$\text{Maximize } a_1(q_0 - q_3) + a_2(q_0 - q_4)$$
$$\text{subject to } a_1 + a_2 \leq C_0.$$

The solution is

$$(a_1, a_2) = \begin{cases} (C_0, 0), & \text{if } q_0 - q_3 \geq q_0 - q_4 \text{ and } q_0 - q_3 > 0 \\ (0, C_0), & \text{if } q_0 - q_4 > q_0 - q_3 \text{ and } q_0 - q_4 > 0 \\ (0, 0), & \text{if } q_0 \leq q_3 \text{ and } q_0 \leq q_4. \end{cases}$$

Similarly, we find that

$$(a_{11}, a_{10}) = \begin{cases} (C_1, 0), & \text{if } q_1 \geq q_3 \\ (0, C_1), & \text{if } q_3 > q_1 \end{cases}$$

and

$$(a_{22}, a_{20}) = \begin{cases} (C_2, 0), & \text{if } q_2 \geq q_4 \\ (0, C_2), & \text{if } q_4 > q_2. \end{cases}$$

Thus, each link serves the buffer with the maximum backpressure, if that backpressure is positive. For instance, it may happen that link 0 stops serving packets. In this network, the routing decisions at link 0 are to route to the shortest queue $q_3$ or $q_4$, provided that this shortest queue is shorter than $q_0$. One may object to this solution that packets from connection 0 will arrive out of order at the destination, which may cause TCP to retransmit packets unnecessarily. This problem can be solved by a reordering buffer at the destination with a suitable timeout scheme.

Note that the decisions of where to send packets and which packets to serve are made packet by packet. This contrasts with quasi-static routing and scheduling decisions. In today's network, one configures scheduling decisions such as weighted fair queueing or deficit round robin by setting parameters. Similarly, the routing decisions are precomputed and downloaded in forwarding tables that determine the next hop for each flow. One may worry about possible oscillations when dynamic routing is used. However, in this design, the decisions are based on the backlog in the next buffer, so that the delays in the control loops are very short. This also contrasts with attempts at dynamic routing in overlay networks that adjust the routing based on observed average delays over overlay paths, a scheme that is potentially unstable, as is well explained in [1].

## 4. Scheduling

In congestion control, one adjusts the share of links that different connections get. Abstractly, this is the allocation of infinitely divisible resources. Scheduling, on the other hand, concerns the allocation of indivisible resources. Thus, instead of sharing a pie, one decides who gets the whole pie. For instance, if $N$ nodes want to transmit but they interfere in a way that only subsets of them can transmit simultaneously, the problem is to decide which subset should transmit at any given time. Most such indivisible allocation problems are NP-hard because of their combinatorial nature. To avoid this computational intractability, one relaxes the problem by considering randomized allocations that work well on average. The adjustment of the allocation probabilities is continuous and one may hope for tractable algorithms. For a detailed presentation of these ideas, see [8].

We illustrate the approach on the simple situation sketched in Fig. 8. Two users are competing for a single resource that they need to perform tasks. The problem is to determine which user should get the resource at any given time. The goal is to maximize the usefulness of the resource,

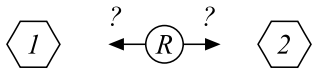


**Fig. 8** Two users 1 and 2 compete for one resource $R$. When the resource become available, the system decides whether to allocate it to user 1 or 2.

measured by the utility that it provides the users. To be more precise, say that user $i$ requires the resource for some random time that has mean $1/\mu_i$ to perform one task, for $i = 1, 2$.

We formulate the problem as follows:

$$\text{Maximize } U_1(x_1) + U_2(x_2)$$
$$\text{subject to feasibility.}$$

In this formulation, $x_i$ is the rate at which user $i$ *works* on tasks and $U_i(x_i)$ is a concave increasing function. Note that $x_i$ is not the completion rate of tasks, but the rate of work on tasks.

Since only one user can work on tasks at any one time, we see that

$$x_1 + x_2 \leq 1.$$

These conditions characterize the *feasible* rates $x = (x_1, x_2)$. In a centralized setting, one could solve the constrained optimization problem and determine the rates $x^* = (x_1^*, x_2^*)$ that solve it. One could then allocate the resource to user 1 with probability $p$ and to user 2 with probability $1 - p$ whenever it becomes available. With this scheme, user 1 would get to keep the resource a fraction $p\mu_1^{-1}/(p\mu_1^{-1} + (1-p)\mu_2^{-1})$ of the time. Accordingly, one would calculate $p$ so that this fraction is $x_1^*$.

However, we are looking for a feedback scheme that adapts automatically to the values of $\mu_1$ and $\mu_2$ that may not be known exactly. We think of each user placing requests at rate $x_i$ in a counter that decrements at rate one whenever user $i$ has the resource. Thus, these counters keep track of the *deficit* between the users' desired and achieved resource possession times. Let $q_i$ be the value of the counter of user $i$. For $i = 1, 2$, define $\pi(i)$ as the fraction of time that user $i$ has the resource and $\pi(0)$ as the fraction of time that no user has the resource. Thus, the counter $q_i$ increases at rate $x_i$ and decreases at rate $\pi(i)$.

We can reformulate the problem in terms of $\pi = (\pi(0), \pi(1), \pi(2))$ as follows:

$$\text{Maximize } U_1(x_1) + U_2(x_2)$$
$$\text{subject to } x_1 \leq \pi(1), x_2 \leq \pi(2),$$
$$\text{and } \pi(1) + \pi(2) \leq 1.$$

For the purpose of deriving our algorithm, we modify this problem as follows:

$$\text{Maximize}$$
$$\sum_{i=1,2} U_i(x_i) + \beta\{H(\pi) - \sum_{i=1,2} \log(\mu_i)\pi(i)\}$$
$$\text{subject to } x_1 \leq \pi(1), x_2 \leq \pi(2),$$

and $\pi(1) + \pi(2) \leq 1$

where $\beta$ is a positive constant and

$$H(\pi) = -\sum_{i=0}^{2} \pi(i) \log(\pi(i))$$

is the *entropy* of the allocation probabilities $\pi$. As we explain below, the term with coefficient $\beta$ allows us to derive a distributed algorithm. Note also that the term between curly brackets is bounded for all $\pi$. Hence, by choosing $\beta$ small, the change in the objective function can be made negligible.

As before, we replace this problem by that of maximizing the objective function minus a multiple of the drift of the sum of the squares of the counter values.

Since

$$\frac{d}{dt} q_i^2(t) = 2q_i(t)[x_i - \pi(i)],$$

we want to maximize

$$\sum_{i=1,2} U_i(x_i) + \beta\{H(\pi) - \sum_{i=1,2} \log(\mu_i)\pi(i)\} - \rho \sum_{i=1,2} q_i[x_i - \pi(i)] \quad (4)$$

subject to

$$\pi(0) + \pi(1) + \pi(2) = 1. \quad (5)$$

To maximize (4) with respect to $x_i$, we find that, for $i = 1, 2$,

$x_i$ maximizes $U_i(x_i) - \rho q_i x_i$.

Thus, in a way similar to the congestion control scheme, the users maximize their net utility when they pay a price $\rho q_i$ for each new request that they place for the resource. One can view $\rho q_i$ as a *congestion price*.

Finally, one maximizes (4) with respect to $\pi$, subject to (5). To do this, one replaces $\pi(0)$ by $1 - \pi(1) - \pi(2)$ and one considers the maximization of

$$\beta H(\pi) + \sum_{i=1,2} \pi(i)\alpha_i \quad (6)$$

where $\alpha_i = \rho q_i - \beta \log(\mu_i)$. We set to zero the partial derivative of (6) with respect to $\pi(i)$, for $i = 1, 2$ and we find

$$0 = -\beta - \beta \log(\pi(i)) + \beta + \beta \log(\pi(0)) + \alpha_i,$$

so that

$$\pi(i) = \pi(0)\mu_i^{-1} \exp\{\gamma q_i\}, i = 1, 2$$

where $\gamma = \rho/\beta$. Since $\pi(0) + \pi(1) + \pi(2) = 1$, one has

$$\pi(0) = \left[1 + \sum_{i=1,2} \mu_i^{-1} \exp\{\gamma q_i\}\right]^{-1}.$$

The next step is to design an algorithm that implements these probabilities $\pi(i)$. One can verify that $\pi$ is the invariant distribution of the continuous-time Markov chain with the transition diagram shown in Fig. 9. One implementation of this Markov chain is that user $i$ requests the resource with rate $\exp\{\gamma q_i\}$ where $q_i$ is the value of the counter of user $i$. When he has the resource, user $i$ releases it after an exponential time with rate $\mu_i$. It should be noted that this state transition diagram is *insensitive*, so that the users get the resources with the fraction of times $\pi(i)$ even if the resource holding times are not exponentially distributed. Observe also that the invariant distribution assumes that the request rates remain constant, although they do not since the backlogs change as users place new requests and release the resource. Nevertheless, one can show that this scheme essentially achieves the maximum sum of utilities. (Technically, one shows that the Markov chain converges fast enough compared to the changes in backlogs.)

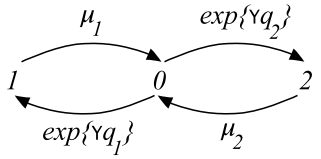


**Fig. 9** Markov chain with invariant distribution $\pi$.

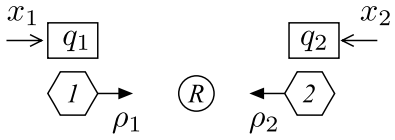


**Fig. 10** Illustration of the resource scheduling algorithm. Each rate $x_i$ maximizes $U_i(x) - \alpha q_i x$. User $i$ requests the resource with a rate $\rho_i := \exp\{\gamma q_i\}$.

Summarizing, the algorithm is as follows and is illustrated in Fig. 10:

- each user $i$ maintains a counter $q_i$ that tracks the deficit between the number of tasks he would like to perform and those he actually performs;
- user $i$ places new requests in that counter with a rate $x_i$ that maximizes $U_i(x_i) - \alpha q_i x_i$ where $q_i$ is the value of the counter of user $i$ and $\alpha$ is a positive constant;
- when it does not have the resource, user $i$ requests it with a rate equal to $\exp\{\gamma q_i\}$ where $\gamma$ is a positive constant.

This algorithm can be contrasted with the binary exponential backoff. Here, instead of having a randomized delay that increases with the number of collisions that a packet has experienced, the delay is computed based on the deficit between the target number of tasks and the actual performed tasks. If the tasks are packet transmissions, then this deficit is the actual backlog of packets in the buffer of the link. The same basic algorithm applies to networks where tasks must be processed by a sequence of processors that compete for resources.

## 5. Conclusion

The last two decades have seen a considerable amount of work by many researchers on the topics of this tutorial. One key idea was to focus on stability instead of trying to minimize average delays, which is an intractable problem except in systems with a lot of symmetry. This focus started with [17] that introduced the idea of minimizing the drift of the sum of the squares of the queue lengths and showed that the resulting control stabilizes the queues if they can be stabilized, i.e., if the arrival rates are *feasible*.

The approach introduced in [9], [10] attacks the utility maximization directly. By solving the corresponding constrained optimization problem using the dual problem and a gradient projection algorithm, the problem is decomposed into individual rate adjustments for the users who maximize their net utility and the buffers that present a shadow price proportional to their backlog. A similar algorithm is achieved by combining the drift of the sum of the squares of the queue length with the utility of the flows, as shown in [15] and [16]. Also, the analysis of this algorithm enables to obtain lower bounds on the performance that reveal the tradeoff between throughput and delay. This optimization approach enables also to derive backpressure and routing algorithms.

The analysis of indivisible resource allocations follows a similar approach to congestion control. The main difference is to tune the probabilistic allocation by tracking the deficit between a desired allocation and the actual allocation. Each user can add to his desired allocation, but at a price that increases with the current deficit. Also, each user requests the resources with an urgency that increases with his deficit.

Summing up, the algorithms have a very intuitive economic interpretation in terms of shadow prices that acts as congestion signals. In some systems, these shadow prices could be actual prices. Such schemes might be appropriate for new services, such as cloud services, processing systems, or specialized networks.

## Acknowledgments

This tutorial was written while the authors were supported in part by NSF-NetSE grants 1024318 and 0910702 that included a collaboration funded by NICT. They acknowledge the fruitful discussions with Dr. Longbo Huang of Tsinghua University, Takeshi Kubo of KDDI Labs in Tokyo, Professors Masaki Aida of Tokyo Metropolitan University and Chisa Takano from Hiroshima City University, Stephan Adams, Vijay Kamble, and Ramtin Pedarsani from U.C. Berkeley.

### References

[1] D.P. Bertsekas and R.G. Gallager, Data Networks, 2nd ed., Prentice-Hall, 1992.

[2] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," IEEE J. Sel. Areas Commun., vol.18, no.3, pp.535–547, 2000.

[3] S. Boyd and L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.

[4] V. Cerf and B. Kahn, "Towards protocols for internetwork communication," IFIP/TC6.1, NIC 18764, INWG 39, Sept. 1973.

[5] D.-M. Chiu and R. Jain, "Networks with a connectionless network layer, part iii: Analysis of the increase and decrease algorithms," Tech. Rep. DEC-TR-509, Digital Equipment Corporation, Stanford, CA, Aug. 1987.

[6] V. Jacobson, "Congestion avoidance and control," Proc. SIGCOMM'88, ACM, Stanford, CA, Aug. 1988.

[7] L. Jiang and J. Walrand, "A distributed CSMA algorithm for throughput and utility maximization in wireless networks," 46th Annual Allerton Conference on Communication, Control, and Computing, Sept. 2008.

[8] L. Jiang and J. Walrand, Scheduling and Congestion Control for Wireless and Processing Networks, Morgan-Claypool 2010.

[9] F.P. Kelly, "Charging and rate control for elastic traffic," European Transactions on Telecommunications, vol.8, pp.33–37, 1997.

[10] F.P. Kelly, A. Maulloo, and D. Tan, "Rate control for communication networks: Shadow prices, proportional fairness and stability," Journal of the Operational Research Society, vol.49, no.3, pp.237–252, 1998.

[11] X. Lin, N.B. Shroff, and R. Srikant, "A tutorial on cross-layer optimization in wireless networks," IEEE J. Sel. Areas Commun., vol.24, no.8, pp.1452–1463, Aug. 2006.

[12] S.H. Low and E. Lapsley, "Optimization flow control, I: Basic algorithm and convergence," IEEE/ACM Trans. Netw., 1999.

[13] R. Metcalfe, "Packet communication," MIT Project MAC Technical Report MAC TR-114, Dec. 1973.

[14] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," IEEE/ACM Trans. Netw., vol.8, no.5, pp.556–567, Oct. 2000.

[15] M.J. Neely, E. Modiano, and C.P. Li, "Fairness and optimal stochastic control for heterogeneous networks," Proc. IEEE Infocom, 2005.

[16] M. Neely, Stochastic Network Optimization with Application to Communication and Queueing Systems, Morgan-Claypool, 2010.

[17] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks," IEEE Trans. Autom. Control, vol.37, no.12, pp.1936–1949, Dec. 1992.

[18] A. Parekh and J. Walrand, Sharing Network Resources, Morgan-Claypool, 2013.

**Jean Walrand** received his Ph.D. in EECS from UC Berkeley and has been on the faculty of that department since 1982. He is the author of *An Introduction to Queueing Networks* (Prentice Hall, 1988) and *Communication Networks: A First Course* (2nd ed. McGraw-Hill,1998) and co-author of *High-Performance Communication Networks* (2nd ed, Morgan Kaufman, 2000), *Communication Networks: A Concise Introduction* (Morgan & Claypool, 2010), *Scheduling and Congestion Control for Communication and Processing networks* (Morgan & Claypool, 2010) and *Sharing Network Resources* (Morgan & Claypool, 2013). His research interests include stochastic processes, queuing theory, communication networks, game theory and the economics of the Internet. Prof. Walrand is a Fellow of the Belgian American Education Foundation and of the IEEE and a recipient of the INFORMS Lanchester Prize, the IEEE Stephen O. Rice Prize, the IEEE Kobayashi Award and the ACM Sigmetrics Achievement Award.

**Abhay K. Parekh** received a B.E.S. in the Mathematical Sciences from Johns Hopkins University, a S.M. in Operations Research from the Sloan School of Management, MIT, and a Ph.D. in Electrical Engineering and Computer Science from MIT in February, 1992. He was a Member of Technical Staff at AT&T Bell Laboratories from 10/85-6/87, and a Postdoctoral Fellow at the Laboratory for Computer Science at MIT between 2/92-6/92. In 10/92, he joined the High Performance Computing and Communications group at IBM as a Scientific Staff Member. In 1995 he moved to Sun Microsystems where was Principal Architect for Network Services. In 1998 he co-founded and was CEO of FastForward Networks. He has been a venture partner at Accel Partners and CEO of several startup companies. He is currently an Adjunct Professor of EECS at U.C. Berkeley. His current research interests are in wireless and video distribution networks, as well as in more general systems involving distributed protocols. While a student at MIT, Dr. Parekh was a Vinton Hayes Fellow and a Center for Intelligent Control Fellow. He is co-author of *Sharing Network Resources* (Morgan & Claypool, 2013). A paper from his Ph.D. dissertation, jointly authored with Prof. Robert Gallager, won the INFOCOM'93 best paper award. Another paper from his dissertation won the William R. Bennett Prize Paper Award in 1994.