Job Scheduling and Multiple Access

Sibi Raj, Emre Telatar, and David Tse

ABSTRACT. We establish an analogy between the additive Gaussian multiple access channel and multi-processor queues. We exploit this analogy to investigate certain problems arising in multiple access information theory. In particular we address the question of minimal average delay for multiple users, each with a finite bit pool, and then a multiple access channel with users arriving according to a Poisson process. In the latter case, we show that the system is stable under all arrival rates, and show that the communication analog of shorter-tasks-faster strategy is close to optimal for high arrival rates.

1. Introduction

A multiple access communication system is comprised of a set of transmitters sending information to a common receiver. Transmitters are driven by independent information sources generating streams of packets; typically this is a bursty stream. Each transmitter encodes the packets it receives in a signal, and the receiver observes a noisy signal that depends on these transmitted signals. The transmitters are uncoordinated except via a "weak" feedback they may get from the receiver.

Information theory views multiple access as a matter of combating noise and interference. The issue of bursty packet arrival is something to be dealt by appropriate source coding. Network theory, on the other hand, views multiple access as a matter of resource allocation and distributed scheduling, taking advantage of the bursty arrival of packets, but ignoring the issues of noise and interference. Neither approach is entirely satisfactory [**Ga**]: By dropping the question of source burtiness, the information theoretic approach makes any meaningful analysis of end-to-end delay impossible, by trivializing the issue of interference and noise, the networking approach takes an oversimplified view of the physical layer.

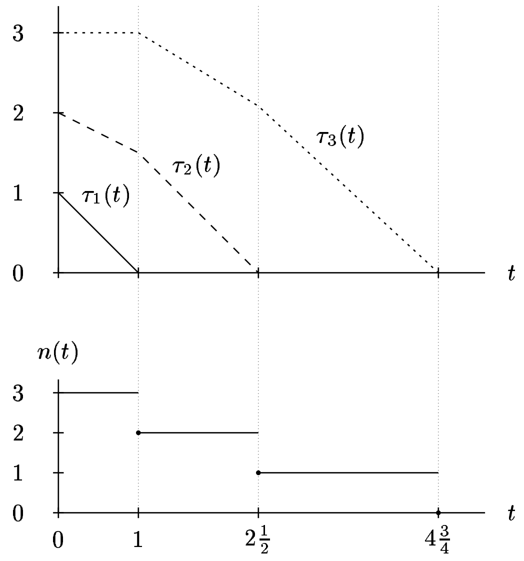
Our aim here is to present a partially unified view. We will consider a Gaussian multiple access channel with equal power transmitters, and will attempt to say something about average packet delay while keeping to an information theoretic

1991 *Mathematics Subject Classification.* Primary 94A24, 68M20; Secondary 94A40, 90B36.
Key words and phrases. Packet delay, network information theory.
S. Raj is supported in part by the NCCR-MICS, a center supported by the Swiss National Science Foundation under grant number 5005-67322.
Part of this work was performed in 1995 while both E. Telatar and D. Tse were with AT&T Bell Laboratories, Murray Hill, NJ.

framework. Our method is based on the exploitation of a certain analogy between this channel and job scheduling on multi-processor queues. Accordingly, first, we will discuss scheduling.

2. Job Scheduling

Suppose we have n tasks to complete. Task k requires τ_k units of service. To complete these tasks we have m processors; processor k can deliver service at a rate of s_k units of service per unit time. We can assign tasks to processors, and are able to change the assignment at any time. However, at any given time no more than one processor may be assigned to any task. Nor can any processor be assigned to more than one task at any given time. A task is completed when the cumulative service it has received meets its service requirement. We will call such a system a multi-processor queue.



The top graph shows the remaining service requirement $\tau_k(t)$ of each task as a function of elapsed time t. The lower graph shows the number of tasks $n(t)$ still to be completed, again as a function of the elapsed time.

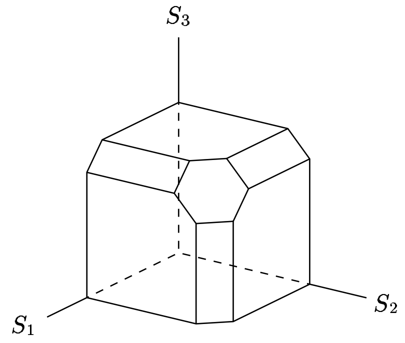
FIGURE 1. Job Scheduling for Example 2.1

EXAMPLE 2.1. Let there be three tasks and two processors ($n = 3$, $m = 2$). Let $\tau_1 = 1$, $\tau_2 = 2$, $\tau_3 = 3$, and $s_1 = 1$, $s_2 = \frac{1}{2}$. We may assign processor 1 to task 1 and processor 2 to task 2 for one unit of time. At the end of this period, task 1 will be completed and task 2 will have a remaining service requirement of $1\frac{1}{2}$ units of service. We can then assign processor 1 to task 2 and processor 2 to task 3 for $1\frac{1}{2}$ units of time. At the end of this period, task 2 will be completed, and task 3 will have $2\frac{1}{4}$ units of service remaining. We can now assign processor 1 to task 3 for $2\frac{1}{4}$ units of time and all tasks will be complete. (See Figure 1.)

2.1. Service Rate Vectors. Suppose we have an infinite collection of processors, with service rates $s_1, s_2, \ldots$. The case of a finite number m of processors can be included in this framework by defining the service rates of all but the first m processors to be zero. Without loss of generality, assume that $s_1 \geq s_2 \geq \cdots$. Suppose we have n tasks at hand. Clearly, we only need to consider processors 1 through n, since at any given time we cannot use more than n processors and these are the n most powerful processors we have.

By assigning processor k to task k we can deliver a service rate vector of $(s_1, \ldots, s_n)$, where the k^{th} component of this vector is the service rate delivered to the k^{th} task.

By permuting the assignment, for any permutation π of the integers $\{1, \ldots, n\}$, the service rate vector $s_\pi = (s_{\pi(1)}, \ldots, s_{\pi(n)})$ is achievable.



The vertices of the hexagon are those rates achievable by permuting the assignments of tasks to processors. The hexagon itself is achieved by time sharing between its vertices. Any point in the entire region is dominated by some point in the hexagon.

FIGURE 2. Rate region example with $n = 3$.

By moving between these assignments arbitrarily quickly, any convex combination of the rate vectors s_π can be achieved. It is also clear that any rate vector which is dominated in every component by an achievable rate vector is also achievable. Thus the achievable service rate vectors form a polymatroid (see e.g., [**W**, §18.3–4]), and thus are those $(S_1, \ldots, S_n)$ that satisfy

$$\text{for all } I \subset \{1, \ldots, n\}, \qquad \sum_{i \in I} S_i \leq \sum_{k=1}^{|I|} s_k \tag{2.1}$$

(see Figure 2). That one cannot achieve any other service rate vector follows from the form of (2.1): if any of the inequalities is not satisfied, then there is a collection I of tasks which get service at a rate exceeding the sum rate of the $|I|$ fastest processors, a contradiction.

2.2. Service Policies. Given a collection of processors and a set of tasks, the state of the system at time t is completely determined by the remaining service requirement $\tau_k(t)$ of each task k at this time. We will denote the number of uncompleted tasks at this time by $n(t)$, that is, $n(t) = |\{k : \tau_k(t) > 0\}|$.

A service policy P is a function that maps a remaining service requirement vector to an achievable service rate vector. Thus, a policy determines the rate of service offered to each task based on the state of the system.

EXAMPLE 2.2. Suppose there is an infinite number of processors, with service rates $s_1, s_2, \dots$ and that $s_1 \geq s_2 \geq \cdots$. Consider a policy L that assigns faster processors to the shorter tasks. If the state of the system is $(\tau_1, \dots, \tau_n)$ and if π is the permutation of the integers $\{1, \dots, n\}$ that orders the tasks in increasing order of service requirement, i.e.,

$$\tau_{\pi(1)} \leq \cdots \leq \tau_{\pi(n)}$$

then, the policy L will assign to $(\tau_1, \dots, \tau_n)$ the service rate vector

$$(s_{\pi^{-1}(1)}, \dots, s_{\pi^{-1}(n)}).$$

We will call L the "shorter tasks faster" policy. Note that in Example 2.1 we employed this policy.

2.3. Scheduling for the Smallest Sum Completion Time. Again suppose we have an infinite number of processors with rates $s_1, s_2, \dots$ with $s_1 \geq s_2 \geq \cdots$, and n tasks with service requirements $\tau_1, \dots, \tau_n$ with $\tau_1 \leq \cdots \leq \tau_n$. Let $C_k(\tau_1, \dots, \tau_n; P)$ denote the completion time of task k under a given service policy P. The quantity

$$\bar{C}(\tau_1, \dots, \tau_n; P) = \frac{1}{n} \sum_{k=1}^{n} C_k(\tau_1, \dots, \tau_n; P)$$

is then the average time the tasks spend in the system

We are interested in finding a policy P which minimizes $\bar{C}$. It is clear that the minimization of $\bar{C}$ is equivalent to the minimization of $\sum_k C_k$, the sum completion time. Fortunately for us, this is a well studied problem in scheduling theory. In the scheduling terminology the multi-processor system we have described is known as "uniform machines with preemption" (the adjective "uniform" indicating that the processors differ only in speed), and the following is well known [**Go**] (see, e.g., [**BEP**, p. 171]):

THEOREM 2.3. *Policy L of serving shorter tasks faster minimizes the average completion time.*

REMARK 2.4. Policy L not only minimizes $\sum_{k=1}^{n} C_k$, it also minimizes $\sum_{k=1}^{j} C_k$ for every $j = 1, \dots, n$.

REMARK 2.5. Under policy L, since shorter tasks are served faster, if $\tau_i < \tau_j$, then $\tau_i(t) < \tau_j(t)$ for all time t: there is no 'overtaking.'

REMARK 2.6. Under policy L, if $\tau_i < \tau_j$, then increasing the length of task j does not change C_i: longer tasks do not interfere with the service of shorter ones.

3. The Gaussian Multiple access Channel

We now turn to the Gaussian mutiple access, and show that results in scheduling theory can be adapted to questions arising in its treatment. Suppose we have n

transmitters with equal power P and a single receiver. The signal received at the receiver is the sum of the signals of the transmitters and Gaussian noise:

$$y = \sum_{k=1}^{n} x_k + z \tag{3.1}$$

where y is the received signal, x_k is the signal transmitter by the k^{th} transmitter, and z is a zero mean Gaussian random variable with unit variance. Suppose that transmitter k has a data rate of R_k. Call a data rate vector $(R_1, \dots, R_n)$ achievable if for any prescribed error probability, no matter how small, there exists codes with rates arbitrarily close to the given data rate for each transmitter and a decoding strategy for the receiver such that the error probability is smaller than that prescribed. It is well known (see e.g., [**CT**, §14.3]) that a data rate vector $(R_1, \dots, R_n)$ is achievable if and only if[1]

$$\text{for all } I \subset \{1, \dots, n\}, \qquad \sum_{i \in I} R_i \leq \frac{1}{2}\log(1 + |I|P). \tag{3.2}$$

For $k = 1, 2, \dots,$ let

$$\begin{aligned} s_k &= \frac{1}{2}\log(1 + kP) - \frac{1}{2}\log(1 + (k-1)P) \\ &= \frac{1}{2}\log\left(1 + \frac{P}{1 + (k-1)P}\right). \end{aligned} \tag{3.3}$$

Observe that s_k is non-increasing in k and that for $m = 0, 1, \dots,$

$$\frac{1}{2}\log(1 + mP) = \sum_{k=1}^{m} s_k.$$

Thus, equation (3.2) can be expressed as

$$\text{for all } I \subset \{1, \dots, n\}, \qquad \sum_{i \in I} R_i \leq \sum_{k=1}^{|I|} s_k. \tag{3.4}$$

Comparing equations (2.1) and (3.4) we see that the data rates achievable over an additive Gaussian multiple access channel are the same as the service rates available from a multi-processor server with processor rates given by (3.3). This suggests an analogy between multiple access channels and multi-processor queues. To illustrate the analogy consider the following problem.

3.1. Transmitters with finite bit pools. Suppose there are n users of a Gaussian multiple access system, each user with power P and user k having a message of length τ_k bits to transmit. Note that unlike the usual information theoretic scenario the users do not have a stream of data arriving at a given rate, but each has a fixed pool of bits to send. We want to minimize the average delay of the messages, the delay of a message being the time until the message is decoded by the receiver. From the previous section, we know how to solve this optimization problem: Assume without loss of generality that $\tau_1 \leq \cdots \leq \tau_n$. Operate at the rate vector $(s_1, \dots, s_n)$ until the message of user 1 is decoded. By this time a number of bits of the remaining messages will be decoded also. Next operate at the rate vector $(0, s_1, \dots, s_{n-1})$ until the message of the second user is decoded.

[1]We measure data rates in this paper in bits, and the logarithms are taken to be base 2.

Continue in this fashion until all the messages are decoded. The resulting decoding times $C_k(\tau_1, \ldots, \tau_n; L)$ will have the least sum (and arithmetic mean) of all possible decoding schemes. Let $\Sigma(\tau_1, \ldots, \tau_n) = \sum_k C_k(\tau_1, \ldots, \tau_n; L)$ denote this least sum.

From an information theoretic point of view, it is clear that no encoding and decoding scheme can have an average delay less than the one computed via this analogy and policy L; there will not be enough mutual information and Fano's inequality will establish a lower bound on the probability of error. However, it is not clear if the average delay as computed by the analogy can actually be achieved: since the number of bits to be transmitted is finite, the usual machinery of the law of large numbers cannot be brought onto this problem. One can, however, look at the case of each user having a large number of bits.

THEOREM 3.1. *Suppose user k has $\lceil \alpha\tau_k \rceil$ bits to transmit ($k = 1, \ldots, n$), where $\alpha > 0$ is a scaling parameter. Then the sum completion time, when normalized by α, can be made arbitrarily close to $\Sigma(\tau_1, \ldots, \tau_n)$ by choosing α sufficiently large.*

PROOF. We will show that for any $\epsilon > 0$, there exists a large enough α and an encoding and decoding scheme such that the error probability is no more than ϵ and the normalized sum completion time is within ϵ of $\Sigma(\tau_1, \ldots, \tau_n)$.

For $0 \le \delta < 1$, let $\Sigma^{(\delta)}(\tau_1, \ldots, \tau_n)$ denote the sum completion time of the tasks when the processor rates are reduced by a factor δ, i.e., when the processor rates are $(1-\delta)s_1, (1-\delta)s_2, \ldots$. Note that $\Sigma^{(\delta)} = \Sigma/(1-\delta)$ is a continuous function of δ. Given $\epsilon > 0$, first choose $\delta > 0$ such that

$$\Sigma^{\delta}(\tau_1, \ldots, \tau_n) < \Sigma(\tau_1, \ldots, \tau_n) + \epsilon.$$

Now consider a multi-processor queue with these reduced rates, and let $C_1, \ldots, C_n$ denote the completion times of the tasks, and let $t_1 = C_1$, $t_k = C_k - C_{k-1}$. There are thus n epochs, $[0, t_1)$, $[t_1, t_2)$, $\ldots$, $[t_{n-1}, t_n)$ during which the assignment of the processors to tasks are fixed. During the j^{th} epoch, processor 1 is assigned to task j, processor 2 to task $j+1$, etc. Let τ_k^j denote the amount of work done on task k during the j^{th} epoch, $\tau_k^j = t_j(1-\delta)s_{k-j+1}$, $1 \le j \le k \le n$. Choose now β large enough so that for all $1 \le j \le k \le n$,

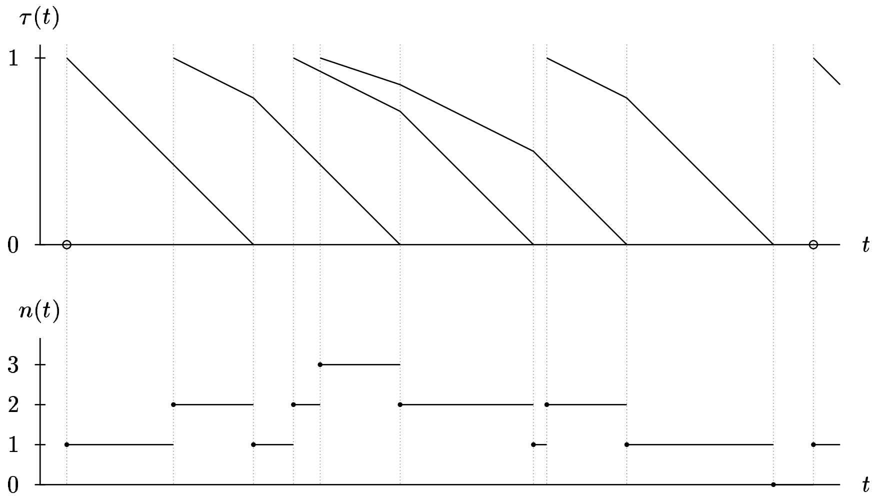
$$\frac{\lceil \beta\tau_k^j \rceil}{\lfloor \beta t_j \rfloor} < s_{k-j+1}.$$

We can now choose a positive integer M such that there exists n multiple access codes, the j^{th} of blocklength $M\lfloor \beta t_j \rfloor$ and rate vector

$$\left(\frac{\lceil \beta\tau_k^j \rceil}{\lfloor \beta t_j \rfloor} : k = j, \ldots, n \right)$$

and each with error probability less than ϵ/n. Choosing $\alpha = M\beta$, we thus have a coding scheme with error probability at most ϵ and normalized sum completion time within ϵ of $\Sigma(\tau_1, \ldots, \tau_n)$. $\square$

3.2. Poisson Arrivals to a Gaussian Multi-user System. Suppose that tasks are not all present at the beginning of time, but they arrive one by one. In the language of scheduling theory, the tasks now has 'release dates,' furthermore the policy needs to be 'on line'. Under these circumstances, it is known that the policy L of serving shorter tasks faster does not necessarily minimize the average time tasks spend in the system. Indeed, policy L is not necessarily optimal even in the case when each arriving task demands 1 unit of service.



A sample path for a multi-processor queue with tasks arriving according to a renewal process. Here $s_k = 1/k$, $k = 1, 2, \ldots$, and each task requires 1 unit of service. The upper curve traces the remaining service requirement of each task as they arrive to the queue. Note that if there are n tasks in the system, the service rate each receives is determined by the temporal ordering of their arrivals; the k^{th} task receives $1/k$ units of service per unit time. The renewal instants for the system, i.e., the instants of arrival to an empty system is marked with a circle in the upper time axis.

FIGURE 3. Sample path with tasks arriving over time

Nonetheless, it is interesting to examine how policy L behaves in this last situation. Since the service demand of a new arrival is 1, it is necessarily at least as large as the service demands of the tasks already in the system. Thus, if there are n tasks in the system just before an arrival, the new task is assigned the $n+1^{\text{st}}$ processor. It also follows from Remark 2.6 that the completion time of a task is independent of the arrival times of the tasks arriving later. Thus, policy L operates in a "first come fastest served" fashion (see Figure 3). To further illustrate the idea, one may think of the tasks lining up in front of the processors. A new arrival goes to the end of the line and moves to faster processors as tasks before it are completed, until it reaches the fastest processor and departs when its service is finished.

Although we remarked that policy L is not optimal in general, it is possible that it does minimize the expected time tasks spend in the system for some particular arrival processes. In any case, we can reach some conclusions without requiring the optimality of policy L. For example, suppose that the arrival process is a renewal process with rate λ, and that $\sum_{k=1}^{\infty} s_k$ diverges. Then, the overall system will be stable for all finite λ. To see this, consider the amount of work that needs to be done to clear the system. This quantity increases by 1 every time a new task arrives, and decreases at a rate of $\sum_{k=1}^{n} s_k$ when there are n tasks to complete. Since customers arrive at a rate of $1/\lambda$, the rate at which the new work arrives is $1/\lambda$, and the rate at which work is done will exceed the arrival rate when n is large enough. The remaining work thus performs a random walk with negative drift and a boundary at the origin, and hence will enter a compact set around the origin infinitely often.

We can now consider a continuous time Gaussian multiple access channel of single-sided bandwidth W. Let all the transmitters have the same power, and let P be the ratio of a transmitters power to the power of the additive Gaussian noise.

At a time when there are n active transmitters, the capacity region of this channel is again given by (3.4) with

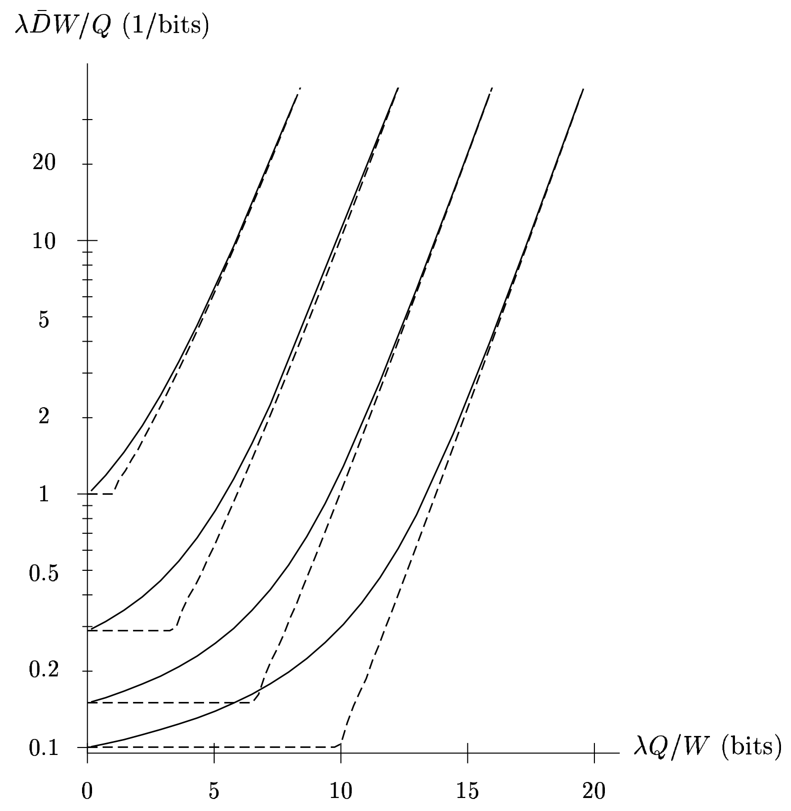
$$s_k = W \log\Bigl(1 + \frac{P}{1 + (k-1)P}\Bigr). \tag{3.5}$$

Now suppose that the packets arrive to this system according to a Poisson process. Just as in the analysis of the ALOHA protocol, we will assume that each arriving packet is associated to a different transmitter. This simplifies the analysis, but is less innocent than it looks. In the case of ALOHA this assumption is a pessimistic one, here it is not so: on the one hand it disallows the cooperative scheduling of packets that would have arrived to the same user, on the other hand, by creating more users it increases the total available transmitted power. It is thus neither an optimistic nor a pessimistic assumption.

Suppose each arriving packet has a fixed number Q of bits to be transmitted. We wish to process the packets in analogy with policy L: if an arriving packet finds an empty system it should be transmitted at the rate s_1; if it finds a system with one other packet, it should transmitted at rate s_2 until the first user is decoded, after which it should be transmitted at rate s_1 until it itself is decoded. In general, an arriving packet that finds $j-1$ packets already in the system should go through transmission rates $s_j, \dots, s_1$ in that order with the transitions occurring at the times when the packets before it are decoded. Note that these times are independent of the later arrivals.

Of course, it is one thing to wish to serve the arriving packets according to a policy which prescribes what rates they should be served at what times, and another thing to be able to come up with a encoding and decoding method that can implement these rates. It is not true that an arbitrary job scheduling policy can be turned into a communication scheme. The remarkable property of policy L is that it is implementable with a small amount of feedback from the receiver. The key enabler is again Remark 2.6: when a new transmission starts, the receiver can inform the transmitter of the decoding times of the transmissions already in progress. The transmitter then knows at which data rates it should communicate at which periods of time, and also when it is to be decoded. Since the transmission rates it sends at are precisely those which allow it to be decoded regarding the existing transmissions as noise, it can be decoded, *and its signal can be subtracted from the received signal, causing no interference to the transmissions scheduled to complete before it.*

Let us illustrate this through an example. Suppose a user finds an empty system. It will then use a code with rate $s_1 = W\log(1+P)$, which gives it a decoding time of $C_1 = S/s_1$. Suppose a second user arrives at a time $r_1 < C_1$. It is not clear, a priori, that the first user can still be decoded at time C_1 in the presence of the new user. To see how this can be done, split the second user's bits into two parts: a first part consisting of $(C_1 - r_1)s_2$ bits and a second part consisting of the remaining bits. For the duration from r_1 to C_1, let the second user transmit at rate $s_2 = W\log\bigl(1 + \frac{P}{1+P}\bigr)$. At the end of this duration, the first part of the bits of the second user can be decoded by regarding the first user as noise. Once these bits are known, the second user's signal can be canceled from the received signal and



The average delay (solid curves) of a packet at different SNR's versus the arrival rate under policy L. Note that the vertical axis is delay normalized by the bandwidth and also the packet length Q (in bits), and the horizontal axis is the arrival rate (in bits per unit time) normalized by the bandwidth. The curves are for SNR values of $0, 10, 20$ and 30 dBs. Also shown (dashed curves) is a lower bound on average delay for any policy.

FIGURE 4. Average Delay vs. Arrival rate

the first user can be decoded as well. The second user will now transmit the rest of its bits at rate s_1. Note that s_k is precisely the rate up to which we can decode a user in the presence of $k-1$ others by regarding these other users as noise.

We should note that the discussion above is valid only when Q is relatively large; otherwise, the splitting of the bits may result in the number of messages to be distinguished during a time interval becoming fractional, or the code lengths becoming too short to allow allow efficient encoding. It should be clear, nonetheless, that an argument of the type used in Theorem 3.1 will establish the required asymptotic. A perhaps cleaner argument will use error exponents as done in [**TG**] and obtain the results above as a limiting case.

Figure 4 shows the average delay $\bar{D}$ of a message (i.e., the time elapsed since its arrival till its decoding) as a function of the arrival rate and the signal to noise ratio P. Note the normalizations of the axes in this figure; the horizontal axis is the bit arrival rate per unit bandwidth, and the vertical axes is the delay per bit in units of inverse bandwidth.

Since we do not claim any optimality of policy L, what guarantee is there that there is not some other, more clever, policy that makes the average delay $\bar{D}$ much less than that achievable by L? The next section derives a lower bound to this average delay and indicates that policy L cannot be beaten by much.

3.3. A lower bound on average delay. We will now derive a lower bound on the average delay for any arrival process of rate λ and any service policy. To that end, suppose we have a multiprocessor queue to which unit length tasks arrive at a rate of λ. Suppose we have some service policy, and let p_n denote the long term fraction of time during which there are n tasks in the queue. Thus, the time-average number of tasks in queue is $\bar{N} = \sum_n np_n$. By Little's law, the average time $\bar{D}$ a tasks spends in the system satisfies

$$\lambda\bar{D} = \bar{N}. \tag{3.6}$$

Observe now, that since the total service rate cannot exceed $\sigma_n = \sum_{k=1}^n s_k$ when there are n tasks in the queue, the long-term average total service rate offered by the system is at most

$$\sum_n p_n\sigma_n.$$

For stability we need $\lambda \le \sum_n p_n\sigma_n$, and thus,

$$\bar{N} \ge \inf\Big\{\sum_n np_n : \sum_n p_n\sigma_n \ge \lambda\Big\},$$

or equivalently,

$$\begin{aligned}\lambda &\le \sup\Big\{\sum_n p_n\sigma_n : \sum_n np_n \le \bar{N}\Big\}\\ &= \sup\{E[\sigma_N] : E[N] \le \bar{N}\},\end{aligned}$$

where the supremum is over all non-negative integer valued random variables N. Let $\sigma : [0,\infty) \to [0,\infty)$ be obtained by linearly interpolating of σ_n; $\sigma(x) = \sigma_{\lfloor x\rfloor} + (x - \lfloor x\rfloor)[\sigma_{\lceil x\rceil} - \sigma_{\lfloor x\rfloor}]$. Since $\{s_n : n \ge 1\}$ is a non-increasing, non-negative sequence, σ is a concave, non-decreasing function. Thus, by relaxing the integer restriction on N, and using Jensen's inequality

$$\begin{aligned}\lambda &\le \sup\{E[\sigma(X)] : E[X] \le \bar{N}\}\\ &\le \sup\{\sigma(E[X]) : E[X] \le \bar{N}\}\\ &\le \sigma(\bar{N}).\end{aligned}$$

Combined with Little's law (3.6) this yields

$$D \ge \sigma^{-1}(\lambda)/\lambda, \tag{3.7}$$

where σ^{-1} is the inverse function of σ. One can check that for an arrival process with constant interarrival times this lower bound is achieved by policy L, so the bound cannot be improved without assuming something more about the arrival process.

Figure 4 shows this lower bound as the dashed curves. It is seen to be tight at high arrival rates, the bound is essentially indistinguishable to the performance of policy L. We thus conclude that policy L, even if it is not optimal, performs quite well in this regime.

4. Remarks

The reader will have noticed that this correspondence is based on the fact that both the multi-processor queue and the multiple access channel are governed by the same formal equation (eqs. (2.1) and (3.4)). We exploit this relationship to adapt problems that are natural in one to the other, and thus obtain novel results in the field of multiple access.

A genuine understanding of multiple access communication requires a combined treatment of bursty sources and noisy channels. Network information theory is difficult enough without further complicating it, but we feel that it is still possible to incorporate some aspects of bursty sources. The approach outlined here is perhaps a little fragile—the results we derive are heavily dependent on, for example, the symmetry between the transmitters—nonetheless, perhaps it can serve as an encouragement for further research.

Acknowledgments

E. Telatar and D. Tse would like to thank S. Hanly, H.-L. Huynh and A. Orlitsky for numerous comments and discussions.

References

[BEP] J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt and J. Węglarz, *Scheduling computer and manufacturing processes,* Springer, Berlin, 1996.

[CT] T. Cover and J. Thomas, *Elements of information theory,* Wiley, New York, 1991.

[Ga] R. G. Gallager, "A perspective on multiaccess channels," *IEEE Transactions on Information Theory,* vol. IT-31, no. 2, pp. 124–142, Mar. 1985.

[Go] T. Gonzalez, "Optimal mean finish time preemptive schedules," Technical report 220, Computer Science Department, Pennsylvania State University, 1977.

[TG] İ. E. Telatar and R. G. Gallager, "Combining queueing theory and information theory for multiaccess," *IEEE Journal on Selected Areas in Communication,* vol. 13, no. 6, pp. 963–969, Aug. 1995.

[W] D. J. A. Welsh, *Matroid theory,* Academic Press, London, 1976.

Laboratory of Information Theory, École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland

E-mail address: `bsraj@lth.epfl.ch`

Laboratory of Information Theory, École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland

E-mail address: `Emre.Telatar@epfl.ch`

Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley CA 94720, USA

E-mail address: `dtse@eecs.berkeley.edu`