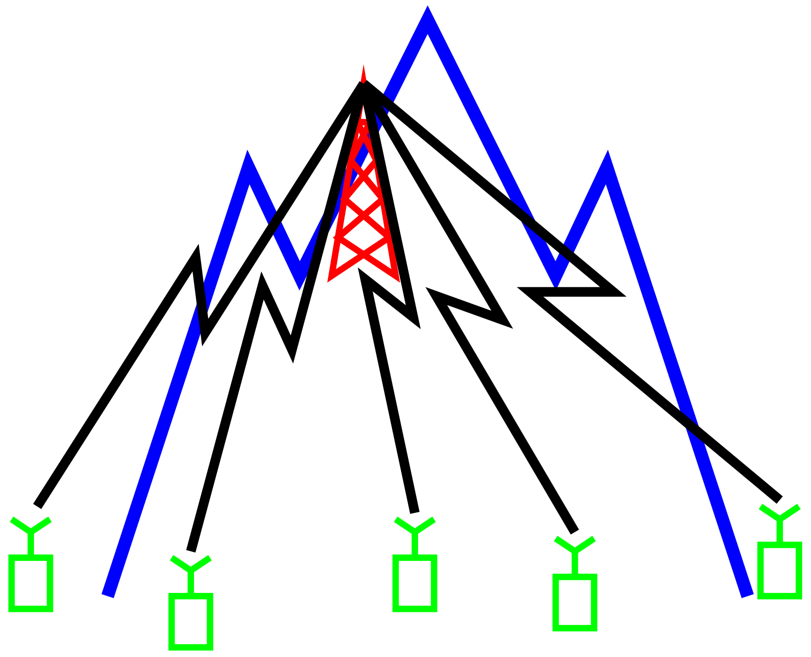
Job Scheduling and Multiple Access

Emre Telatar, EPFL

Sibi Raj (EPFL), David Tse (UC Berkeley)

Multiple Access Setting



Characteristics of Multiple Access:

- Bursty Arrivals
- Uncoordinated Transmitters
- Interference
- Limited Feedback

Different bodies of research approach these characteristics in different ways.

Multi Processor Queues

Consider n tasks to be completed. Task k requires s_k units of service.

We also have m processors. Processor j can deliver service at a rate r_j units of service per unit time.

We can:

- Assign processors to tasks
- Change the assignment at any time

cannot:

- Assign > 1 processor to a task at a given time
- Assign > 1 task to a processor at a given time

Task k is complete when the cumulative service it has received equals s_k.

Multi Processor Queues

Example:

We have several cows,

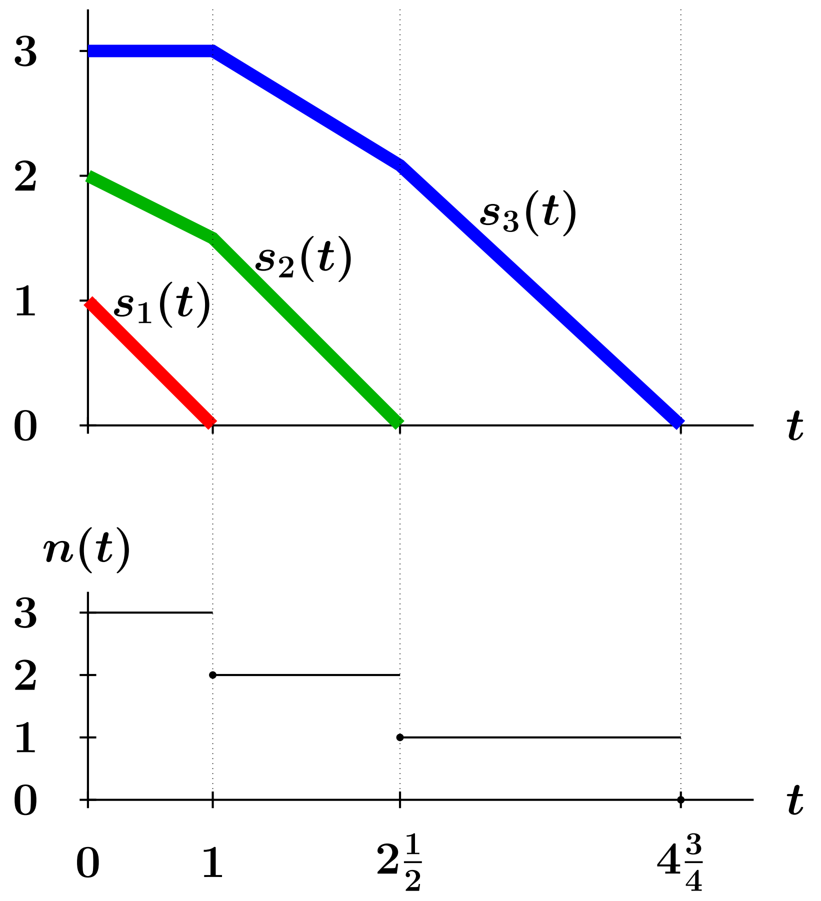
```
         (__)
         (oo)
  /-------\/
 / |     ||
*  ||----||
   ~~    ~~
```

$$s_k = \text{amount of milk in cow } k$$

and we have a number of milkers to milk them

$$r_j = \text{amount of milk milker } j \text{ can milk in an hour}$$

Example



Three tasks with

$$s_1 = 1, \quad s_2 = 2, \quad s_3 = 3.$$

Two processors with

$$r_1 = 1, \quad r_2 = 1/2.$$

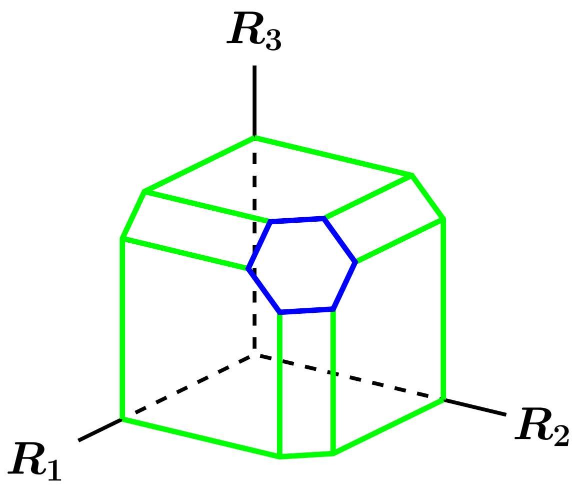
First assign the processor 1 to task 1 and processor 2 to task 2. When task 1 completes, assign processor 1 to task 2, processor 2 to task 3. When task 2 completes, assign processor 1 to task 3.

Service Rates

Assume

$$r_1 \geq r_2 \geq \cdots$$

If we have only three tasks to serve, what is the set of service rates (R_1, R_2, R_3) that they can receive?



The vertices of the hexagon and those rates achievable by permuting the assignment of the first three processors to the three tasks. The hexagon itself can be achieved by time sharing, and any point contained in the polymatroid is dominated by some point of the hexagon. The set of service rates are then given by

$$R_i \leq r_1, \quad R_i + R_j \leq r_1 + r_2 \quad (i \neq j), \quad R_1 + R_2 + R_3 \leq r_1 + r_2 + r_3$$

Service Rates

In general, if we had n tasks, the deliverable service rates are those $(R_1, \ldots, R_n)$ that satisfy:

$$\boxed{\text{for every } I \subset \{1, \ldots, n\}, \qquad \sum_{i \in I} R_i \leq \sum_{j=1}^{|I|} r_j.}$$

That is, for any collection of tasks, the total rate these tasks receive cannot exceed the the total rate they would get if the fastest processors were assigned to them.

Service Policies

Given a collection of processors and a set of tasks, if we know the remaining service requirement $s_k(t)$ for each task k at a time t, we do not need to know anything else about the past of the system to determine the future evolution. That is,

$$\left(s_k(t), k = 1, 2, \dots\right)$$

is a state of the system.

A service policy is a function that maps the state of the system to a deliverable service rate vector. It thus determines the rate of service offered to each task at time t based on the state at time t.

Service Policies (cont)

Example: Assume again $r_1 \geq r_2 \geq \cdots$. Consider a policy that assigns faster processors to shorter tasks. That is: if the state of the system is $(s_1, \ldots, s_n)$ and π is a permutation of $\{1, \ldots, n\}$ that orders the tasks in increasing service requirements, i.e.,

$$s_{\pi(1)} \leq \cdots \leq s_{\pi(n)},$$

then this policy will assign processor i to task $\pi(i)$.

Formally, this policy maps

$$(s_1, \ldots, s_n) \mapsto (r_{\pi^{-1}(1)}, \ldots, r_{\pi^{-1}(n)}),$$

where π is the above permutation.

Job Scheduling

Suppose we serve the tasks using some policy $\boldsymbol{P}$. Let $\boldsymbol{t_k(P)}$ denote the completion time of task $\boldsymbol{k}$.

Define

$$\bar{t}(P) = \frac{1}{n} \sum_{k=1}^{n} t_k(P)$$

as the average completion time of the tasks.

Theorem: The policy of assigning shorter tasks to faster processors minimizes $\bar{t}(P)$. In fact, if $s_1 \leq \cdots \leq s_n$, this policy minimizes $\sum_{k=1}^{j} t_k(P)$ for every $j \leq n$.

Gaussian Multiple Access Channel

We have n transmitters and 1 receiver. The relationship between the transmitted and received signals is

$$Y = x_1 + \cdots + x_n + Z$$

where Y is the received signal
x_k is the signal sent by transmitter k
Z is memoryless Gaussian noise with unit variance

Suppose the transmitters have equal power P. Then the data rate vector $(R_1, \ldots, R_n)$ is achievable if and only if

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

Gaussian Multiple Access Channel

If we define

$$\begin{aligned} r_j &= \tfrac{1}{2}\log\big(1 + jP\big) - \tfrac{1}{2}\log\big(1 + (j-1)P\big), \\ &= \tfrac{1}{2}\log\Big(1 + \frac{P}{1 + (j-1)P}\Big) \end{aligned}$$

then we see that

$$\tfrac{1}{2}\log(1 + |I|P) = \sum_{j=1}^{|I|} r_j,$$

and so the data rate constraints in the achievability condition is

$$\text{for all } I \subset \{1, \dots, n\} \qquad \sum_{i \in I} R_i \le \sum_{j=1}^{|I|} r_j.$$

Thus, the same formula determines the achievable data rates in a multiple access channel and and deliverable service rates in multi processor queues.

Can this be put to some use?

Example Application

Given the number of bits $s_1, s_2, \ldots$ each transmitter has to send—not the data rate, just bits—what transmission strategy will minimize the average transmission time?

Mimic the 'shorter tasks faster' policy: Consider just two users with $s_1 \leq s_2$. Give user 1 service at rate r_1 until its message is decoded at time $t_1 = s_1/r_1$. By then, also decode $t_1 r_2$ bits of the second user. Then, user 2 should get service at rate r_1 until its remaining $s_2 - t_1 r_2$ bits are decoded.

Can this 'wishful analogy' be made into a transmission strategy? Since user 1 is supposed to get rate r_1, it should be as if it is the only user of the channel. But if user 2 is silent, then how is it supposed to get rate r_2? Nevertheless, by a small miracle, all is well. User 2 should break its s_2 bits into two portions of size $t_1 r_2$ and $s_2 - t_1 r_2$. Then, during the interval $[0, t_1]$ we can decode the first portion regarding user 1 as noise, at rate r_2. Once this portion is decoded user 2's signal can be subtracted, and we can indeed decode user 1, at rate r_1.

Multiprocessor Queues with Arrivals

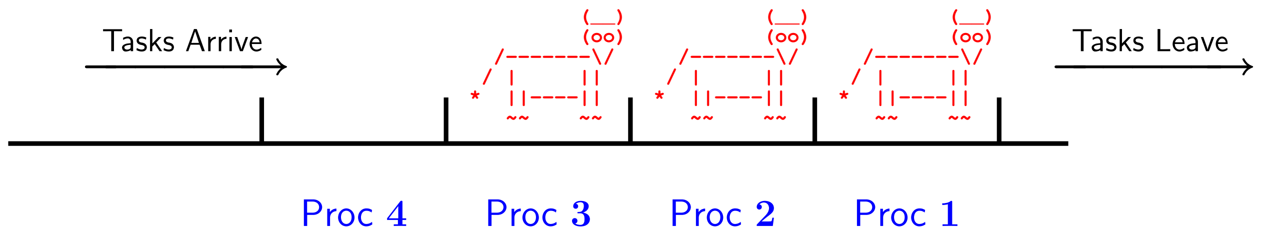
Suppose that the tasks are not all present at the beginning of time, but they arrive at random instants of time. What policy should be used to minimize the expected time a task spends in the system?

I don't know the answer, but we can still propose the 'shorter tasks faster' policy as a heuristic:

Suppose tasks arrive according to some stochastic process (say Poisson) with rate λ. Suppose each task requires 1 unit of service when it arrives.

Observe: under the 'shorter tasks faster' policy a task cannot disturb the service of earlier tasks. Not only is there no 'overtaking', the departure time of a task is known as soon as it arrives, and is independent of all future arrivals.

Multiprocessor Queues with Arrivals



A task finding an empty system is assigned processor 1, otherwise it will be assigned the first idle processor. When the earliest job completes, the remaining tasks are promoted up the processor hierarchy.

One can show that if $\sum_j r_j$ diverges, then the system is stable for any arrival rate λ.

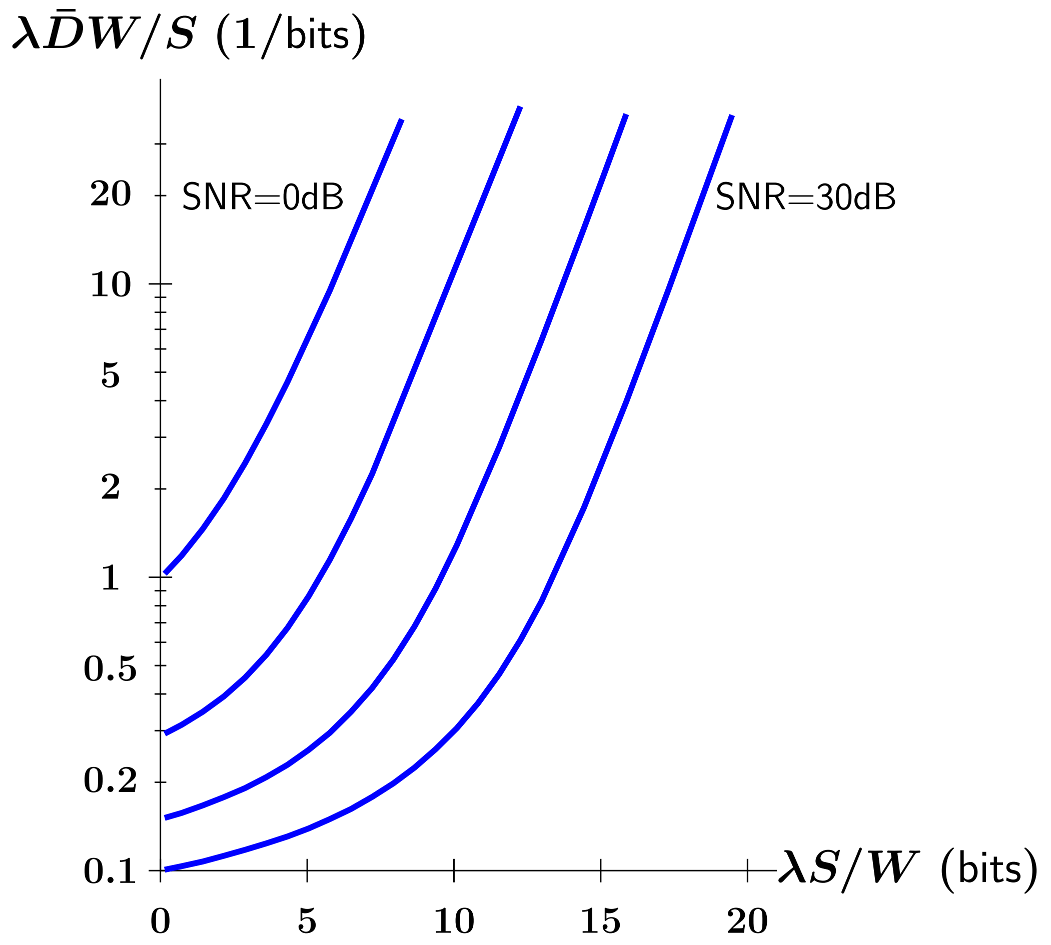
Multiple Access Channel with Arrivals

Can one implement the policy just described as a transmission strategy?

With a bit of feedback, yes: maintain a departures timetable for the transmissions currently in service. An arriving packet that finds n people already in service will need to be served at rates $r_{n+1}, r_n, \ldots, r_1$, the transitions taking place every time some other packet is decoded. With the 'departures timetable' the incoming packet knows when these transitions are to take place, and split itself to subpackets of appropriate size.

Furthermore, during the first interval, the rate it is to get r_{n+1} is exactly what it can get if it is decoded while treating the earlier packets as noise. At the end of the first period, the subpacket can be decoded and subtracted, and thus not disturbing the earlier packets. And similarly for the remaining periods.

Multiple Access Channel with Arrivals



Performance of the policy: average delay vs arrival rate.

A Lower Bound on Delay

Suppose we have a multiprocessor queue to which unit length tasks arrive at a rate λ. Suppose we have some service policy. Let p_n denote the long term fraction of time during which there are n tasks in the queue (assume p_n exists). Thus, the time-average number of tasks in queue is

$$\bar{N} = \sum_{n=0}^{\infty} n p_n.$$

By Little's law, the average time $\bar{D}$ a task spends in the system is given by

$$\lambda \bar{D} = \bar{N}$$

The long-term average service that is offered by the system is at most

$$\sum_{n=0}^{\infty} C_n p_n; \qquad C_n = \sum_{j=1}^{n} r_j$$

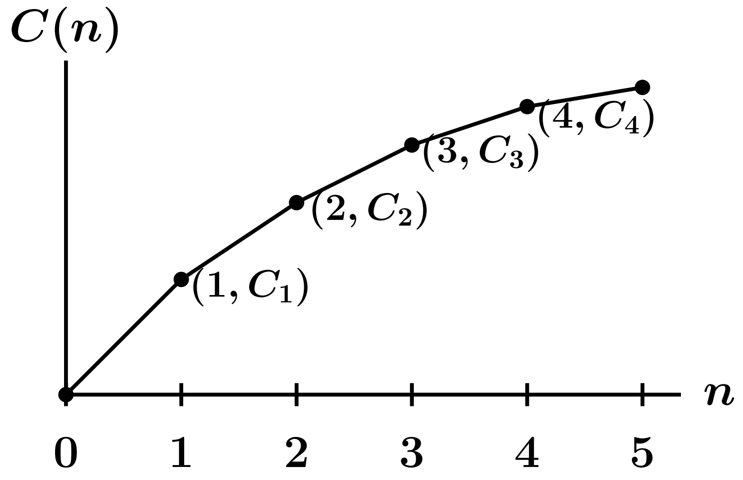
A Lower Bound on Delay

For stability $\lambda \leq \sum_n C_n p_n$, and thus,

$$\bar{N} \geq \inf\left\{\sum_n n p_n : \sum_n C_n p_n \geq \lambda\right\}.$$

Equivalently,

$$\lambda \leq \sup\left\{\sum_n C_n p_n : \sum_n n p_n \leq \bar{N}\right\} = \sup\{E[C(N)] : E[N] \leq \bar{N}\}$$



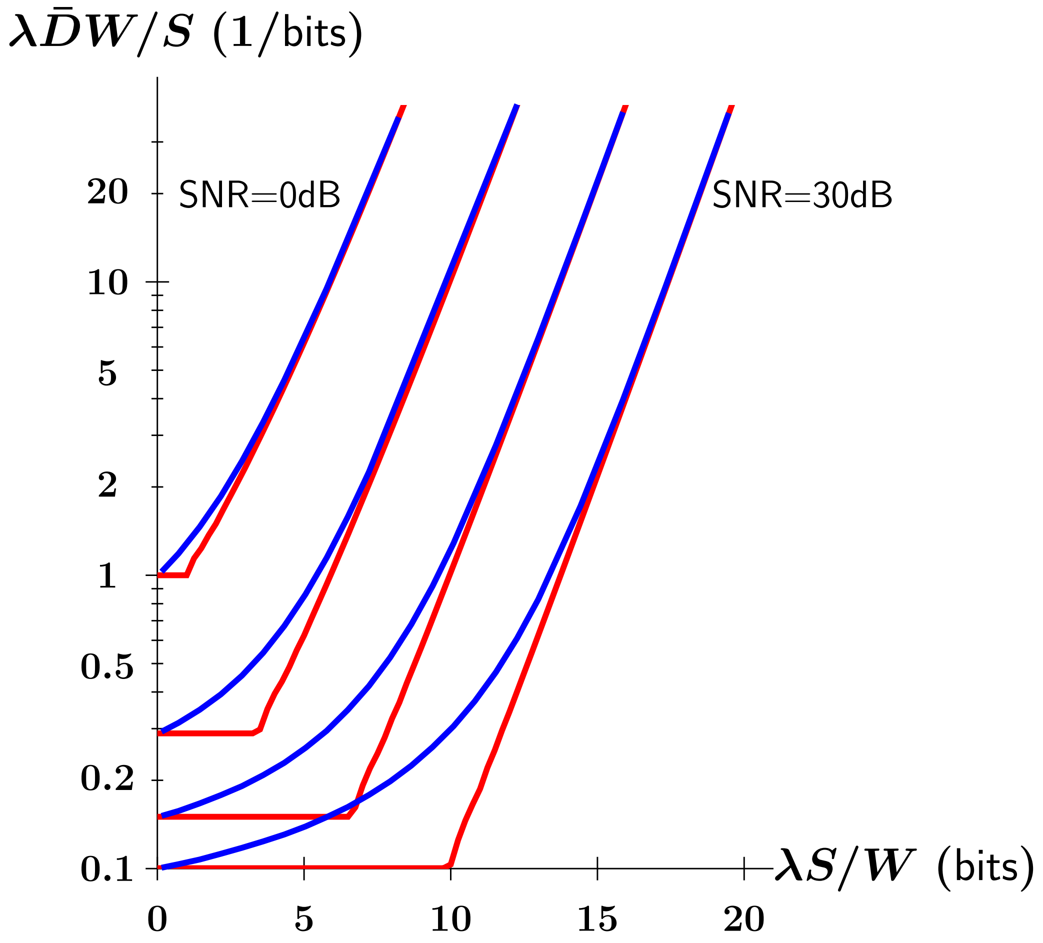
But, by "Jensen's Inequality"

$$E[C(N)] \leq C(E[N]) \leq C(\bar{N}),$$

with equality if p puts mass only on $\lfloor \bar{N} \rfloor$ and $\lceil \bar{N} \rceil$. Thus:

$$\bar{D} \geq \frac{C^{-1}(\lambda)}{\lambda}.$$

Multiple Access With Arrivals



Comparison of the performance of the policy with the lower bound.

Concluding Remarks

Understanding multiuser communications will require a unified view of networking and physical layers.

Network information theory is already hard enough even without trying to incorporate network-layer questions.

Still, we may be able to say more than what we currently do.

The example above is too fragile, but is perhaps useful as an illustration.