In previous lecture we wrote the solution to find optimal line rates to maximize a given utility function. The form of utility function determines if we prefer throughput or fairness.

The problem is that to find optimum, we need global knowledge of the network (all flows on all lines). Instead, we want sources to run their control algorithm based on local information only.

**EXAMPLE** From packet loss & delay I determine congestion and I implement AIMD.

**QUESTION** By doing so am I optimizing the flows over the whole network?

One of the great achievements in network theory in the last decades was to understand the mechanism behind network flow optimization in a distributed fashion.

It requires some math: let's rewrite our problem:

N nodes
M links

Find vector of flows $\mathbf{x}$ over lines:

$$\max_{\mathbf{x}} f(\mathbf{x}) = \sum_{i=1}^{N} U(x_i)$$

s.t.

$$g_j(\mathbf{x}) \leq 0 \quad \forall j = 1 \ldots M$$

where $f(\mathbf{x})$ is concave utility function (remember "law diminishing returns")

$g_j$ are linear flow constraints e.g. $x_1 + x_2 - 1 \leq 0$

So we want $\mathbf{x}^* = \arg \max_{\mathbf{x}} f(\mathbf{x})$

A woful math theorem tells us that this is equivalent to the following dual problem:
find $x^*$ that maximizes $L(x, A^*) = f(x) - \sum_{j=1}^M A_j g_j(x)$

So we want

$$x^* = \underset{x}{\text{arg max}} \ L(x, A^*)$$

This is done in two steps, because first we need to find an optimal value of $A = A^*$ and then optimize over $x$ to find $x^* = x^*$.

First, consider the function of $A$

$$D(A) = \max_x L(x, A), \text{ that is for every } A \text{ there will be an } x(A) \text{ that achieves the maximum,}$$

and find the $A^*$ that minimizes this, namely:

$$A^* = \underset{A}{\text{arg min}} \ D(A)$$

Once you have found this value of $A^*$ plug it back into $x(A)$ to find

$$x^* = \underset{x}{\text{arg max}} \ L(x, A^*)$$

$L$ called Lagrangian

$A_j$ are Lagrangian multipliers or shadow prices

**EXAMPLE** (from previous lecture of flows, 2 arms)

$$L(x, A) = f(x) - \sum_{j=1}^M A_j g_j(x) = u(x_1) + u(x_2) + u(x_3) - A_1 x_1 - A_2 x_2 - \lambda_2 x_1 - \lambda_2 x_3 + A_1 + A_2$$

$$= u(x_1) - (\lambda_1 + A_2) x_1 + u(x_2) - A_1 x_2 + u(x_3) - \lambda_2 x_3 + A_1 + A_2$$

maximize $L(x, A)$ factorizes into 3 disjoint problems:

find $x$

$$\begin{align*}
\text{maximize } & u(x_1) - (\lambda_1 + A_2) x_1 \\
\text{maximize } & u(x_2) - A_1 x_2 \\
\text{maximize } & u(x_3) - \lambda_2 x_3
\end{align*}$$
Since the constraints were linear, each variable $x_i$ appears as a different term and the problem divides into $3$ separate problems that can be solved independently for each flow.

Each user can solve a separate maximization problem to find $D(\lambda)$.

Now we are faced with the problem of finding

$$\lambda^* = \arg \min_{\lambda} D(\lambda) = \arg \min_{\lambda} \max_x L(x, \lambda)$$

To find $\lambda^*$ we use a gradient algorithm.

The algorithm updates $\lambda$ in different steps proceeding on the function $L(x, \lambda)$ along the direction opposite to the gradient (derivative) of the function with respect to $\lambda$. This is the direction of steepest descent with respect to $\lambda$.

For the new value of $\lambda$ the users adjust their rates $x_j(\lambda)$ to maximize $L(x, \lambda)$ and so on.

So we have:

$$\lambda_j(n+1) = \lambda_j(n) - \beta \frac{\partial}{\partial \lambda_j} L(x, \lambda)$$

**Example**

At step $n$ we have:

$$\begin{cases}
    x_1(n) = \arg \max_{x_1} u(x_1) - [\lambda_1(n) + \lambda_2(n)] x_1 \\
    x_2(n) = \arg \max_{x_2} u(x_2) - \lambda_1(n) x_2 \\
    x_3(n) = \arg \max_{x_3} u(x_3) - \lambda_2(n) x_3
\end{cases}$$

These are done locally at the nodes.

Then $\lambda$ is adjusted as:

$$\begin{cases}
    \lambda_1(n+1) = \lambda_1(n) - \beta x_1(n) - \beta x_2(n) \\
    \lambda_2(n+1) = \lambda_2(n) - \beta x_1(n) - \beta x_3(n)
\end{cases}$$

**Question**

How can we do this locally too?
Before answering the question notice the interpretation:

Each link "charges" user a "price" \( p_j \) per unit flow over link \( j \). Therefore each user maximizes the "net utility" for example:

\[
\text{utility of flow } x_j = u(x) - (a_1 + a_2) x_j
\]

So there is a price to pay to use a link because it may cause congestion.

This suggests that the price should be proportional to queue length.

Consider queue length for flow \( j \)

\[
q_j [ (n+1) T] = q_j [ nT] + T g_j (x)
\]

where \( g_j (x) = \frac{x_1 + x_2}{1} \) \( \Rightarrow T g_j (x) \) is the increment in queue length over time \( T \).

Compare (1) with expression for \( q_j (n+1) \)

\[
q_j [ (n+1) T] = q_j [ nT] + p \cdot g_j (x)
\]

If a step gradient algorithm occurs every \( T \) time units of queue evolution, we get:

\[
\frac{\lambda_j}{T} \cdot g_j \Rightarrow \frac{\lambda_j}{T} \cdot q_j [ nT] = \frac{\lambda_j}{T} \cdot g_j (x)
\]

and we can choose the price to evolve as the queue length to have a complete distributed solution.

How to estimate queue length? Measuring delay & losses

When queue increases \( \Rightarrow \) increase the price \( p \) force users to decrease rate

When queue decreases \( \Rightarrow \) decrease the price \( p \) force users to increase rate.

This corresponds to readjust \( x \).