ECE 158 LECTURE 12 Solving the MYSTERY of TCP

In previous lecture we wrote the solution to find optimal link 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 links). 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 $\underline{x}$ over links:

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

s.t.

$$g_j(\underline{x}) \le 0 \quad \forall j = 1 \dots M$$

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

g_j are linear flow constraints

e.g. $x_1 + x_2 - 1 \le 0$

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

A useful math theorem tells us that this is equivalent to the following dual problem:

find x^* that maximizes the Lagrangian $L(\underline{x}, \underline{\lambda}^*) = f(\underline{x}) - \sum \lambda_j^* g_j(\underline{x})$ s.t. $\lambda_j^* g_j(x^*) = 0 \quad \forall j = 1 \dots M$

So we want

$$\boxed{\underline{x}^* = \arg \max_{\underline{x}} L(\underline{x}, \underline{\lambda}^*)}$$

This is done in two steps, because first we need to find an optimal value of $\underline{\lambda} = \underline{\lambda}^*$ and then optimize over $\underline{x}$ to find $\underline{x} = \underline{x}^*$

First, consider the function of $\underline{\lambda}$

$D(\underline{\lambda}) = \max_{\underline{x}} L(\underline{x}, \underline{\lambda})$, (that is for every $\underline{\lambda}$ there will be an $\underline{x}(\underline{\lambda})$ that achieves the maximum)

and find the $\underline{\lambda}^*$ that minimizes this, namely:

$$\underline{\lambda}^* = \arg \min_{\underline{\lambda}} D(\underline{\lambda})$$

Once you have found this value of $\underline{\lambda}^*$ plug it back into $\underline{x}(\underline{\lambda})$ to find $x^* = \arg \max_{\underline{x}} L(\underline{x}\ \underline{\lambda}^*)$.

L called Lagrangian

λ_j are Lagrangian multipliers or shadow prices

EXAMPLE (from previous lecture of 3 flows, 2 links)

$$L(\underline{x}, \underline{\lambda}) = f(\underline{x}) - \sum_{j=1}^{M} \lambda_j g_j(\underline{x}) = u(x_1) + u(x_2) + u(x_3) - \lambda_1 x_1 - \lambda_1 x_2 - \lambda_2 x_1 - \lambda_2 x_3 + \lambda_1 + \lambda_2$$

$$= \underline{u(x_1) - (\lambda_1 + \lambda_2) x_1} + \underline{u(x_2) - \lambda_1 x_2} + \underline{u(x_3) - \lambda_2 x_3} + \lambda_1 + \lambda_2$$

$\max_{\underline{x}} L(\underline{x}, \underline{\lambda})$ factorizes into 3 disjoint problems:

find
- maximize $u(x_1) - (\lambda_1 + \lambda_2) x_1$
- max $u(x_2) - \lambda_1 x_2$
- max $u(x_3) - \lambda_2 x_3$

Since the constraints were linear, each variable x_i appears on a different term and the problem divides into 3 that can be solved <u>independently</u> for each flow.

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

Now we are faced with the problem of finding

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

To find $\underline{\lambda}^*$ we use a <u>gradient algorithm</u>

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

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

So we have:

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

step-size; direction steepest descent opposite to derivative; computing the derivative

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 $\underline{\lambda}$ 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" λ_j per unit flow over link j. Therefore each user maximizes the "net utility", for example $u(x_1) - (\lambda_1 + \lambda_2)x_1$

$(\lambda_1 + \lambda_2)x_1$ → price of flow x_1 that goes through two links.

$u(x_1)$ → utility of flow x_1

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

$$q_j[(n+1)\tau] = q_j[n\tau] + \tau g_j(\underline{x}) \quad (1)$$

where $g_j(\underline{x}) = \underbrace{x_1 + x_2}_{\text{arrival rate}} \underbrace{-1}_{\text{service rate}}$ $\Rightarrow$ $\tau g_j(\underline{x})$ is the increment in queue length over time τ

Compare (1) with expression for $\lambda_j(n+1)$

$$\lambda_j[n+1] = \lambda_j[n] + \beta g_j(\underline{x})$$

β of

If 1 step gradient algorithm acts every τ time units of queue evolution, ~~recalculate~~. To have the same evolution, we let:

$$\lambda_j = \frac{\beta}{\tau} q_j \Rightarrow \lambda_j[n+1] = \underbrace{\frac{\beta}{\tau} q_j[n\tau]}_{\lambda_j[n]} + \underbrace{\frac{\beta}{\tau} \tau g_j(x)}_{\beta g_j(x)} = \frac{\beta}{\tau}\left[q_j(n\tau) + \tau g_j(\underline{x})\right]$$

and we can choose the price to evolve as the queue length ~~of~~ to have a complete distributed solution.
How to estimate queue length? Measuring delay & losses

[When queue increases $\Rightarrow$ increase the price] $\rightsquigarrow$ force users to decrease rate
[When queue decreases $\Rightarrow$ decrease the price] $\rightsquigarrow$ force users to increase rate

this corresponds to readjust $\underline{x}$.