Optimal Transport Directed Study March 3rd, 2023

Optimal Transport Artifact

Author: Ryan Yang *Advisor: Dan Proulx*

Abstract

In this project, we considered a specific problem in optimal transport: the transfer of some item or object, in our case files, along certain links in a network. The internet is based on the seven-layer OSI model, conceptually splitting the task of sending data into seven layers, with the highest layer being the "application layer." However, the application layer has a set of very weak control knobs: the path of the request and the number of active connections on that request. For that reason, human operators cannot easily use the application layer to achieve resource control or optimal throughput. While the field of optimal transport is well-studied in general, the task of optimally assigning integer numbers of TCP connections is not well-studied, with CERN's FTS being the first attempt at varying the number of connections to achieve positive benefits. The main goal of this project is to optimize on this set of control knobs. In this paper, we set out to build a protocol to optimize for the integer assignment of TCP connections that generates the best set of throughput values. To do this, we specifically study the function that takes the connection assignment to empirical throughput values, design algorithms for the optimization of that function, and rigorously analyze and compare different methods both theoretically and empirically.[1]

Outline

- Problem Formulation
- Linear Programming
- FTS and Zero Order Methods

---

1 Problem Formulation

We begin by considering the control knobs available to us at the application layer of the OSI model. We begin by defining what resources are available. Let the network graph be defined as follows:

Definition 1 (Network and Link Capacity)**.** *The network graph (V, L) has the set of vertices V, and the set of links L which each have a total maximum capacity.*

[1] All code can be found at https://github.com/RyanYunruiYang/ConcurrentCongestionControl.

Then, using this resource, the fundamental problem of network transfers is transferring data at source locations to destinations. For ease of analysis, we assume that we can decompose the set of transfers into a set of "pipes."

Definition 2 (Pipe)**.** *A "pipe" is a transfer from a source to a destination, characterized by the set* $\{l_1, l_2, \cdots l_k\}$ *of links that it passes over and the integer value* n_i*, the number of connections open on the pipe.*

Given the setup of the network (V, L), with the link capacities, and the list of pipes, we may now consider the control knob we have: the vector $\vec{n}$, where each value n_i represents the number of connections on pipe i. Then, after the black box TCP layers work to guarantee hardware performance, we are left with the following mapping:

$$\vec{n} \overset{\text{OSI Model}}{\rightarrow} \vec{T} \tag{1}$$

where $\vec{T}$ represents the final total throughput, with T_i being the throughput rate through pipe i in terms of gigabits per second. Note that the implementation of this, as well as the later max-min fairness, can be found in `network.py`.

1.1 Max Min Fairness (MMF)

The actual internet is a highly empirical process with several layers of optimization. For the purposes of analysis, we consider a model of idealized behavior: max-min fairness (MMF). MMF is what will happen if everything at lower levels goes perfectly which, while unrealistic, allows us to analyze application-layer optimization more easily. In general, the definition of Max-Min Fairness is:

Definition 3 (Max-Min Fairness)**.** *A feasible allocation of rates* $\vec{x}$ *is "max-min fair" if and only if an increase of any rate within the domain of feasible allocations must be at the cost of a decrease of some already smaller rate. Formally, for any other feasible allocation* $\vec{y}$*, if* $y_s > x_s$*, then there must exist some* s' *such that* $x_{s'} \leq x_s$ *and* $y_{s'} < x_{s'}$*.*

However, in our case, what MMF means in the context of $\vec{T}$ is that the throughput vector $\vec{T}$ is max-min fair compared to all other possible $\vec{T}$ possible under the given link capacities. Several results are known about MMF, with most of the most powerful results describing the bottleneck structure. Additionally, the waterfilling algorithm, which is proven to be max-min fair, is now often taken as an alternative definition of MMF.

We also introduce Pareto-optimality,

Definition 4 (Pareto Optimality)**.** *A feasible allocation of rates* $\vec{x}$ *is "Pareto-Optimal" if there does not exist some allocation* $\vec{y}$ *where* $y_s > x_s$ *for all* s*. Then, let the Pareto front,* $P(S)$*, be the set of all Pareto efficient solutions of the set* S*.*

Algorithm 1 Waterfilling for Max Min Fairness

1: **Input:** Set of pipes $\mathcal{P}$ and set of links $\mathcal{L}$.
2: **Initialize:** $\vec{T} = 0^{|p|}$, vector of throughputs; set $\mathcal{L}' = \emptyset$ of bottlenecked links; set $\mathcal{P}' = \emptyset$ of bottlenecked pipes
3: **while** $\mathcal{P}' \neq \mathcal{P}$ **do**
4: Find the link l that minimizes the value $\frac{C_l - \sum_{p \in g(l)} T_p}{\sum_{p \in g(l) \cap \mathcal{P}'} n_p}$ where $g(l)$ is the set of pipes passing through link l.
5: For all $p \notin \mathcal{P}'$, increase T_p by $\frac{C_l - \sum_{p \in g(l)} T_p}{\sum_{p \in g(l) \cap \mathcal{P}'} n_p}$.
6: Add l to $\mathcal{L}$.
7: Add all pipes in $g(l)$ to the set $\mathcal{P}'$.
8: **end while**

1.2 Novel MMF Results

As a part of this project, we also work to extend the theory of MMF and have the following new results. The main difference between our analysis here, and the known results in the previous section, is that we care about what the max-min fairness vectors look like as we vary the assignments $\vec{n}$.

To that end, given a specific network, we define an explicit function $f\colon \mathbb{Z}^{|p|} \to \mathbb{R}^{|p|}$, which models the mapping that max-min fairness gives. Now, define two different "feasible" polytopes.

Definition 5 (Capacity-Satisfaction Polytope)**.** *Let the CSV (capacity satisfaction polytope) be the set of throughputs $\vec{T}$ which satisfy the constraints $\sum_{i \in \ell} T_i \leq C_\ell$ where for each link ℓ, the summed over i are the pipes passing through ℓ, and C_ℓ is the capacity of the link.*

Definition 6 (Achievable Polytope)**.** *Let the AP (achievable polytope) be the set of throughputs $\vec{T}$ which are equal to $f(\vec{n})$ for some integer n.*

Based on this, it is clearly true that

$$AP \subseteq \text{Range}(f) \subseteq P(CSV) \tag{2}$$

where Range(f) is the range of f over all positive real number inputs. However, we can strengthen the second inequality to an equality with the following claim and corollary.

Lemma 1. *All Pareto-optimal points in the capacity-satisfaction polytope can be written as $f(\vec{z})$ for some (not necessarily integer) vector $\vec{z} \in \mathbb{R}^{|p|}$*

Proof. Consider the $\vec{x} \in P(CSV)$. It suffices to show that there exists some $\vec{z}$ such that $f(\vec{z}) = \vec{x}$. We in fact claim that any $\vec{z} = \lambda \vec{x}$ will work. To see this, note that in Algorithm 1, the configuration $\vec{z}$ will not be bottlenecked until the water level hits $\frac{1}{\lambda}$. At this water level, the configuration is

temporarily $\frac{1}{\lambda} \cdot \lambda\vec{z} = \vec{z}$. Therefore, we can only go up from here with the rest of the algorithm, giving:

$$f(\lambda\vec{z}) \succeq \vec{z}.$$

However, since $\vec{z} \in P(CSV)$, we also know that $\vec{z} \succeq f(\lambda\vec{z})$. Thus, it must necessarily be true that $f(\lambda\vec{z}) = \vec{z}$ and we are done. □

Theorem 1. *The set of values achievable with max-min fairness can be* ***exactly*** *characterized by the set of rational points in the Pareto front of legal configurations, or formally:*

$$AP = P(CSV) \cap \mathbb{Q}^{|p|} \tag{3}$$

Proof. Firstly, $AP \subseteq P(CSV) \cap \mathbb{Q}^{|p|}$. To see this, firstly $AP \subset CSV$ is obvious. Next, $AP \subset P(CSV)$ since if there existed some $a \in AP, y \in CSV$ such that $y \succ a$, then Algorithm 1 would not have terminated at a. Finally, clearly all final values in AP will be rational numbers, so $AP \subset \mathbb{Q}^{|p|}$.

Now, we turn our attention to proving $P(CSV) \cap \mathbb{Q}^{|p|} \subseteq AP$. To see this, consider $q \in P(CSV) \cap \mathbb{Q}^{|p|}$. Then, using the same argument as Lemma 1.2, we select some integer M such that Mq is an integer, which is possible since q is rational, and then

$$f(Mq) = q$$

and therefore $q \in AP$, and we have shown both directions and have shown both directions of the equality and are done. □

Table 1: Notation

Variable	Meaning
L	set of links
$\vec{n}$	decision vector (number of TCP connections on each pipe)
$\vec{T}$	throughput vector
$P(S)$	Pareto Frontier
CSV	Capacity Satisfaction Polytope
AP	Achievable Polytope

2 Linear Programming (LP)

A natural gut reaction is to design and optimize for some objective function. For example, total throughput, which would give $a(\vec{T}) = \sum_i T_i$, or maximizing the total amount of link resource used, which after a double counting argument is equal to $a(\vec{T}) = \sum_i$ `(length of pipe i)` $\cdot T_i$.

A simple class that captures both of these cases is the following linear form

$$a(\vec{T}) = \sum_{i=1}^{|p|} w_i \cdot T_i \tag{4}$$

where w_i are some set of weights. This can be seen as a linear program (LP), written in standard LP form as:

$$\begin{aligned} &\text{minimize } w^T \vec{x} \\ &\text{subject to } \mathbf{N}\vec{x} \leq \mathbf{C} \\ &\vec{x} \geq 0 \end{aligned}$$

where $\vec{x}$ represents the throughput values, and

$$N_{ji} = \begin{cases} n_i & \text{if pipe } i \text{ is on link } j \\ 0 & \text{else} \end{cases}, \tag{5}$$

which is set up so that the jth entry of $N\vec{x}$ is the usage on link j, and C_j is the capacity on link j. Finally $\vec{x} \geq 0$ since thr throughput must be positive.

2.1 The Structure of Linear Programming in Network Flow Optimization

Definition 7 (Basic Solution)**.** *In a LP where there are m simultaneous linear equations, then a basic solution is a solution with at most m nonzero values.*

Theorem 2 (Fundamental theorem of linear programming)**.** *Given a linear program in standard form where $\boldsymbol{A}$ is an $m \times n$ matrix of rank m, then if there is an optimal feasible solution, there is an optimal basic feasible solution.*

This key observation, the fact that in a linear programming problem with n free variables at most m are used, is what drives the simplex method and allows us to solve LPs quickly. Generally, the simplex method replaces basis vectors out from the set of m currently used until no such swap gives a benefit.

However, this structure, while computationally convenient, also reveals the main problem with optimizing with linear programming. The flip side is that we have $n - m$ pipes with a zero value, which means that optimizing for an LP **completely starves $n - m$ pipes**. Specifically, here n is the number of pipes and m is the number of links, so another way of seeing this is that there is, on average, one active pipe per link. From the perspective of fairness, this is unacceptable, and this sets up the design in Section 3.

2.2 Solving the LP

This specific type of LP is slightly easier to solve because we have a less than or equal to bound. We use the slack trick, introducing a set of slack variables $s_1, s_2, \cdots s_L$ where L is the number of links. Then, if we redefine $\vec{x}$ to be a $L + |p|$ dimensional vector, with the first L values being s_j values, and the next $|p|$ values being T_i values, we have a new LP:

$$\begin{aligned} &\text{minimize } w^T \vec{x} \\ &\text{subject to } \mathbf{N}\vec{x} = \mathbf{C} \\ &\vec{x} \geq 0 \end{aligned}$$

Now, we may set up the simplex method tableau

$$\left[\begin{array}{cc|c} I_L & \mathbf{N} & \mathbf{C} \\ \hline 0_L & \mathbf{w}^T & 0 \end{array}\right] = \left[\begin{array}{c|c} A & C \\ \hline w & 0 \end{array}\right] \tag{6}$$

which is a $(L+1) \times (L + |p| + 1)$ matrix. Then, apply the Simplex Algorithm (implementation can also be found in `simplex.py`).

Algorithm 2 Simplex Method

1: Set up the tableau, with the upper m rows representing conditions and the bottom row w being the payoffs.
2: **while** there exists positive w_j value **do**
3: Select a column q with $w_q > 0$
4: Find the minimum value of $p = \text{argmin}_i \frac{C_i}{A_{iq}}$ over positive A_{iq}.
5: Divide the pth row through by A_{pq}
6: Do row reduction; subtract A_{iq} times the pth row from each of the other rows i (including w, thereby turning the qth column into a basis vector with a 1 at the pth row and 0s everywhere else.
7: **end while**

3 Resource Control Algorithms: FTS and Zero-Order

Current methods for controlling the $\vec{n}$ are limited in scope. The only existing work is CERN's FTS (File Transfer Service). As seen in lines 115 - 152 of the `OptimizerConnections.cpp`, they adjust the decision, which we note as n_i, in response to the average throughput.

However, while this does have valuable intuition – empirically test and move towards methods which work better – it is still working on a single pipe and has not achieved our main goal: using full network information to optimize the decision configuration $\vec{n}$ for the best possible $\vec{T}$.

Algorithm 3 FTS Model Analyzed (Called for High Success Rate)

procedure OPTIMIZEGOODSUCCESSRATE(state)
    if round($\log_B$(`cur.ema`)) < round($\log_B$(`prev.ema`)) **then**
        decision = decision - 1
    else if `cur.ema` $\geq$ `prev.ema` **then**
        decision = decision + 1
    else
        decision = decision
    end if
end procedure

In fact, historically, the CERN FTS algorithm is primarily designed to handle the narrow problem of storage bottlenecks. Without bounds, the general behavior of Algorithm 3 is: connections increase until you reach some sort of deterioration, at which point you oscillate. But, this would not deteriorate for a long time, so in practice, the controller would set a maximum number of connections, which FTS would easily ramp up to, and FTS would almost run at these upper bounds, bringing us back to square one, unsure of how to choose the values for the upper bounds.

3.1 Proposal: Zero Order Methods

Recall that if we let f be the mapping from the decision $\vec{n}$ to throughput $\vec{T}$, then our task is to

$$\max_{\vec{n}\in\mathbb{Z}^{|p|}} a(f(\vec{n})) = \max_{\vec{n}\in\mathbb{Z}^{|p|}} u(\vec{n}) \tag{7}$$

where $u(\cdot) = a(f(\cdot))$ represents the direct mapping from $\vec{n} \to \vec{T} \to a(\vec{T})$ where a is some nonlinear function of $\vec{T}$ (nonlinear to avoid the problems from Section 2).

This has clearly become a more generic problem: minimizing a function over integer inputs. To do this, inspired by FTS, we begin at some estimate n_0, and at each round, we select a random vector $z \in \{\pm e_1, \pm e_2, \cdots \pm e_{|p|}\}$ where e_i is the basis vector with a 1 at i and 0 elsewhere. Then, we compare $u(n_k)$ and $u(n_k + z)$. If $u(n_k + z) > u(n_k)$, then we set $n_{k+1} = n_k + z$. Alternatively, we may estimate the gradient as

$$G_k = \frac{u(n_k + z) - u(n_k)}{\|z\|} = u(n_k + z) - u(n_k) \tag{8}$$

and update n_k in the direction of z proportionally to G.

Implementation of this general idea can be found in increasing levels of complexity in `greedy_search.py`, `zero_order.py`, and `fts4_summer.py`.

3.2 Evaluations/Analysis

Unfortunately, as of March 3rd, 2023, we do not have good empirical or theoretical results. Any suggestions are welcome and should be directed to

y.ryan.yang@gmail.com

References

- Rate adaptation, Congestion Control and Fairness: A Tutorial
- Linear and Nonlinear Programming (Third Edition) by David Luenberger and Yinyu Ye
- CERN FTS3 Code Base, specifically OptimizerConnections.cpp