Traffic Engineering Report

ANDRII IERMOLAIEV, BEN MAYS, ANNE POTTER, and ARON ZHAO

Many companies are currently shifting to software-defined, programmable networks. Network traffic engineering allows us to improve networking costs and performance simultaneously. We will discuss our data management and algorithm implementations. We then analyze the performance of two major networking algorithms to contrast strengths and weaknesses of each algorithm.

CCS Concepts: • **Networks** → **Network design and planning algorithms**.

Additional Key Words and Phrases: Traffic Engineering, Software Defined Networking

ACM Reference Format:
Andrii Iermolaiev, Ben Mays, Anne Potter, and Aron Zhao. 2023. Traffic Engineering Report. 1, 1 (May 2023), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The goal of network traffic engineering is to optimize a network to increase network throughput, decrease latency, increase fault tolerance, decrease operational costs, or a combination of these. An exponential increase in network demand has increased operating costs, which has led to the deployment of traffic engineering techniques to increase network resource use efficiency [2].

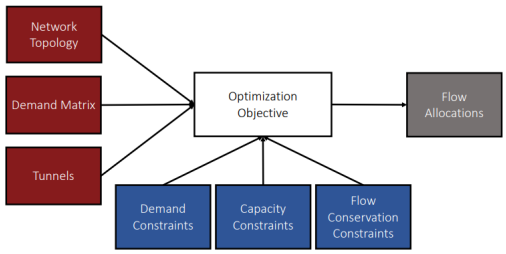

Fig. 1. Visual representation of traffic engineering algorithms. The red boxes are inputs representing topographical features of the network. The blue boxes are constraints placed upon the algorithm to ensure all demands and flow and capacity constraints are met. The gray box represents the solution that is provided when demands are met and objective is achieved. [2]

Authors' address: Andrii Iermolaiev, ai93@cornell.edu; Ben Mays, bm627@cornell.edu; Anne Potter, ap674@cornell.edu; Aron Zhao, az355@cornell.edu.

XXXX-XXXX/2023/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.1 Project Goals

The goal of our project is to utilize software-defined network traffic engineering to optimize the flow of traffic demands across a network. We have implemented two algorithms on the network we were provided: one to maximize the overall throughput, and one to optimize even link usage, minimizing the maximum link utilization. This allows us to compare trade-offs of different optimizations on a network and compare them with regards to reliability and other important network metrics.

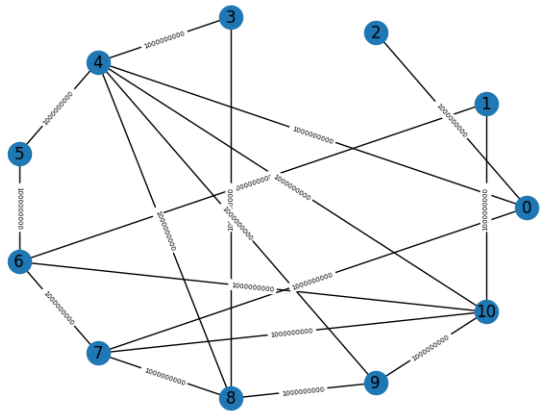

Fig. 2. Diagram of the network we are optimizing transport over provided through node and topology documentation. Each node represents a device on the network. Bandwidth of each connection is printed on the edge.

2 METHODOLOGY

2.1 Network Formulation and Visualization

We began by importing a data set representing a Sprint network from the provided repository. These files provided the links, nodes and demands matrix of that network over a period of time. We used Python and NetworkX to create a graph representing our network topology. We parsed the data from the nodes, edges and demands files and used the NetworkX API to construct the graph. We also used an implementation of DFS to find the k shortest paths from an online source quoted in references [1]. The output, seen in Fig. 2, is made up of numbered nodes representing devices on a network (i.e. switches, servers, etc.), and edges representing links with a corresponding maximum bandwidth that they can provide.

2.2 Throughput Maximization

For each algorithm we created a system of equations to represent the objective and constraints of our system using the Gurobi library API. To maximize the total throughput, we set our Gurobi model objective to be the maximization of the sum of all path flows as seen in Eq 1. This will enable highest throughput since it will push the model to meet all the demands while also loading as many paths as possible with traffic, enabling highest possible throughput in cases where not all demands can be met.

$$max \sum_{t \exists e} f(t) \tag{1}$$

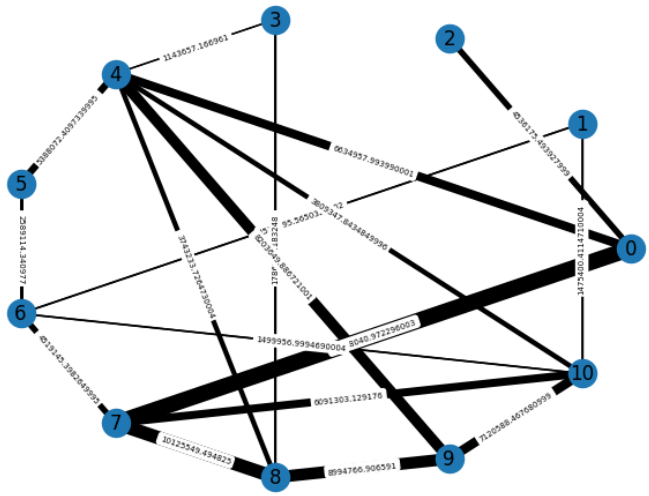

Fig. 3. Results of network optimization for maximum throughput.

In order to achieve these constraints we created these variables within our model in Gurobi:

(1) e0, ..., e17 - representing the values of edge loads within the network.
(2) path0, ..., path541 - representing the values of paths between every single pair of nodes. We opted to use path formulation for our implementation so we chose to only create variables for 5 shortest paths between every pair of nodes. The paths were chosen using the physical number of edges between two nodes.
(3) demandX, Y - representing the total demand between any nodes X and Y.

We then encoded the relationships between the variables using these constraints:

(1) demandX, Y == max_demand([x][y]) - to encode the demand of every demand variable from the maximum demand matrix found within all time stamps. We define the maximum demand matrix to be the one with the highest total value of demands within the matrix of all timestamps.
(2) path_sumX, Y <= demandX, Y - to encode that the sum of path values between nodes X and Y should be less than or equal to the total demand between the two nodes. The total sum of paths should not exceed the total demand so that the trivial solution of just using the total bandwidth of every edge doesn't come up as a viable solution.
(3) path_sumE == eE - to encode that the value of traffic through every edge E is equal to the sum of path values that utilize that edge.
(4) eE <= bandwidthE - to encode that the value of the edge is less than or equal to its maximum bandwidth.

2.3 Link Usage Optimization

To maximize utilization we will minimize the maximum utilization within all the edges. This will incentivize the use of all links to be as close to equal as possible while constraints and demands are

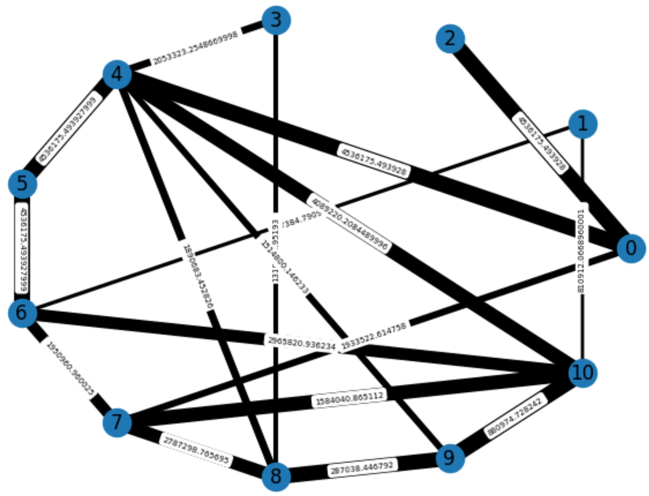

Fig. 4. Results of network optimization for minimum MLU.

still met. Our Gurobi model objective function, seen in Eq 2 minimizes the maximum link usage.

$$min(max(f(t))) \tag{2}$$

For this model we used the same variables with the addition of max_flow variable, which the objective minimized.

All the same constraints were used as in the previous model with one addition and one change:

(1) max_edge == gp.max(edge_vars) - representing that the max edge variable is equal to the largest edge value within the model
(2) path_sumX, Y == demandX, Y, differing from the previous model, in order to encode that the demand must be met and avoid the trivial solution of assigning every edge to zero.

2.4 Gurobi Solve Time Analysis

We conducted an analysis to identify and assess the relationship between network topology size and the Gurobi optimizer solve time for this problem. To mitigate the possible effects of any unidentified characteristic differences in real-world networks of varying sizes, we developed a script to generate networks with specified parameters, allowing for greater control over the independent variable and ensuring greater consistency in all other aspects.
For this analysis, networks were generated with between 5 and 350 nodes. Each network has a density of 0.4, indicating that each node in the network is connected by an edge to approximately 40% of the other nodes. Each edge was assigned a capacity of 1,000 Mbps, and demands were randomly generated between each pair of nodes with an average of 20 Kbps. Running the maximum throughput algorithm on these generated networks yielded the results in Figure 4.

3 RESULTS

Summary of Algorithm Performance			
Algorithm	Maximum Link Usage (%)	Fault Tolerance	Throughput
Maximum Throughput	1.21	Low	Max
Minimum MLU	0.45	High	Max

After analyzing our results we were able to determine that adding a constraint to minimize the MLU makes our traffic allocation more fault tolerant. Maximizing throughput in the first model leads to some edges being significantly more utilized than others, and fewer paths being used between nodes in order to achieve a higher objective value. By forcing the second model to achieve the lowest possible maximum value, we achieve a better spread of traffic to other edges and, hence, achieve higher utilization of shorter paths and more similar link utilization. This means that edges are, on average, less loaded and therefore, in case of failure, less traffic will need to be rerouted than if a link with higher utilization failed. You can see the relative load of the edges by comparing the thicknesses on the traffic allocation diagrams.
The overall throughput in the two models is the same since all the demand has been met. However, in the case that we had lower bandwidth on all the edges the first model would produce a higher value since it would sacrifice some demands being met for the sake of pushing more traffic through the edges.
Figure 4 indicates a clear exponential growth in algorithm solve time with increasing network size.

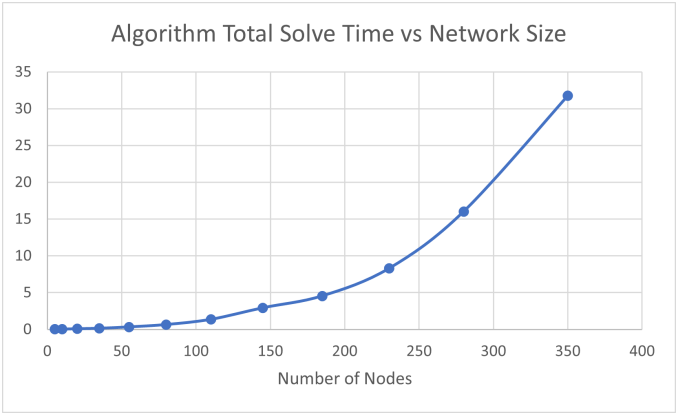

Fig. 5. Graph depicting solve time of maximum throughput algorithm on networks of increasing size.

The y-axis indicates seconds to solve the allocation problem, and the rate of increase grows with increasing number of nodes. A significant driver of this increase is in the network setup, as there is an exponential increase in number of paths being identified and considered in the path-formulation approach to the problem. As contemporary networks often consist of thousands to even millions of nodes, it is clear that software-defined traffic engineering algorithms are likely to be unable to support full traffic engineering over the entire network effectively. This, with the requirement of a thorough setup of the network, which is difficult to maintain in larger networks due to additions, removals, and failures of nodes and links, promotes the use of software traffic engineering on smaller, more controlled networks, for example in internal subsections of a larger network, or in an autonomous system.

REFERENCES

[1] Shivam Gupta. 2023. Print all paths from a given source to a destination. Available at https://www.geeksforgeeks.org/find-paths-given-source-destination/ (2023/05/04).

[2] Rachee Singh, Nikolaj Bjørner, and Umesh Krishnaswamy. 2022. Traffic Engineering: From ISP to Cloud Wide Area Networks. In *Proceedings of the Symposium on SDN Research* (Virtual Event) *(SOSR '22)*. Association for Computing Machinery, New York, NY, USA, 50–58. https://doi.org/10.1145/3563647.3563652