Report: Traffic Engineering

Team: SummerVacationIamComing

Team member: Hung-Han Sung, Kaiyuan Yang, Rui Yang

GitHub Repo: https://github.com/hans990807/CS5456_FinalProject

1. Goals of the project:

The goal of this project is to implement a variety of traffic engineering algorithms and contrast their performance. Before working on the traffic engineering algorithm, we need to understand SDN (Software Defined Network). SDN has one centralized controller machine for the whole network system. Compared to SDN, the traditional network has a control plane and a data plane for each switch, which is highly coupled. Since we can comprehend the whole picture of the network by leveraging SDN, it would be easier to maintain and avoid failure of the network. Additionally, we can write some programs in the centralized control machine to auto-detect suspicious data.

Many cloud services providers like Google, Microsoft, and Amazon use SDN-based routing for their data centers. They have their own goals for their cloud services. So, they design different algorithms to manage the data flows, which is traffic engineering. For example, some companies want to provide their clients with High throughput. Furthermore, some might want to provide their clients with low latency, fault tolerance, or operational cost. As for our project, we first maximize the overall throughput of the network and then minimize the MLU (maximum link utilization).

2. Methodology:

2. 1 Data Processing and Visualization

Before we implement the algorithm. We need to process data and store useful data into data structures. Firstly, we parse the topology and node input file, and visualize the network graph with capacity using networks. Secondly, we parse the traffic demands for later use in algorithm implementation.

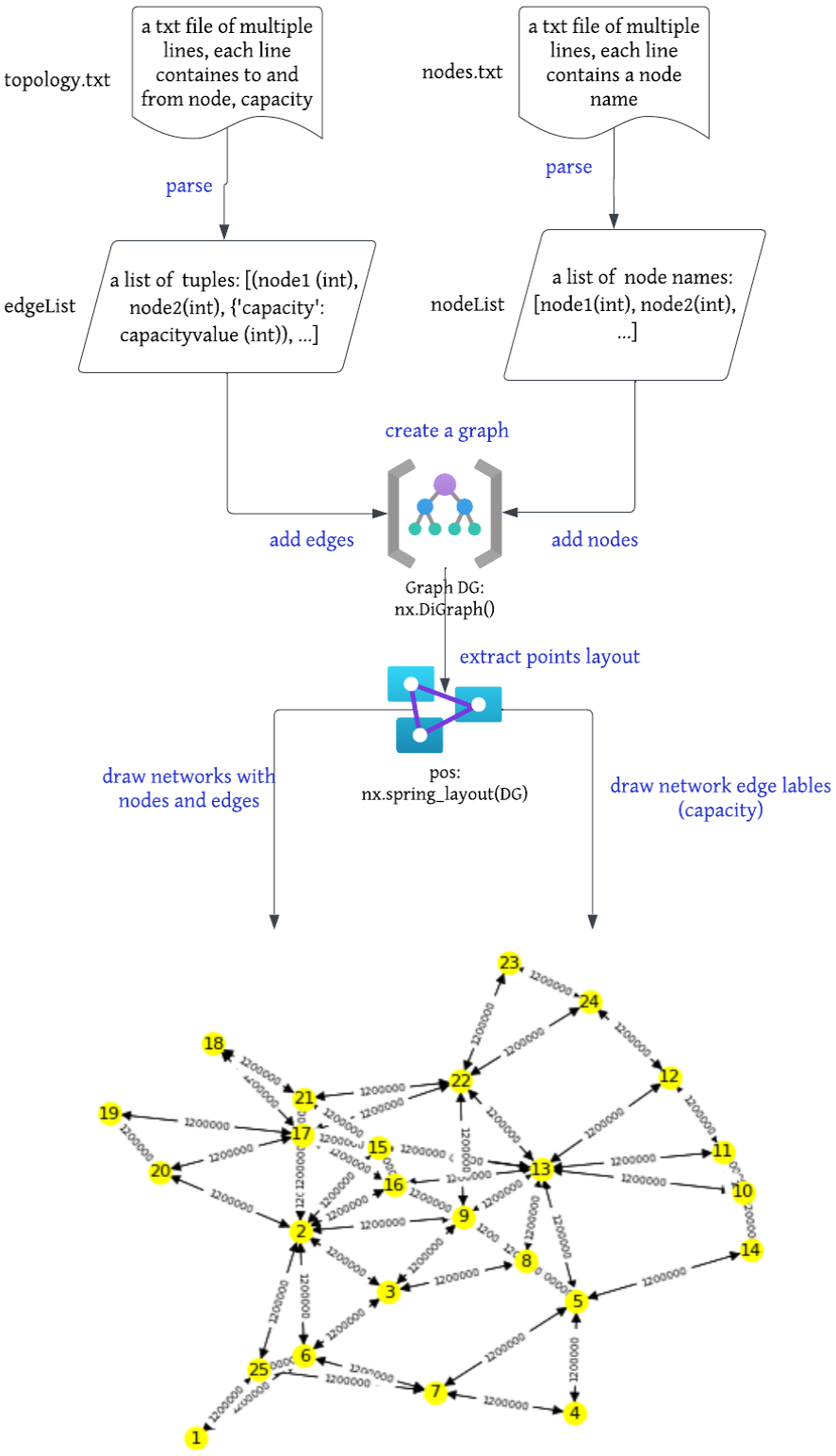

Figure 1 the pipeline of parsing topology and generate network graph.

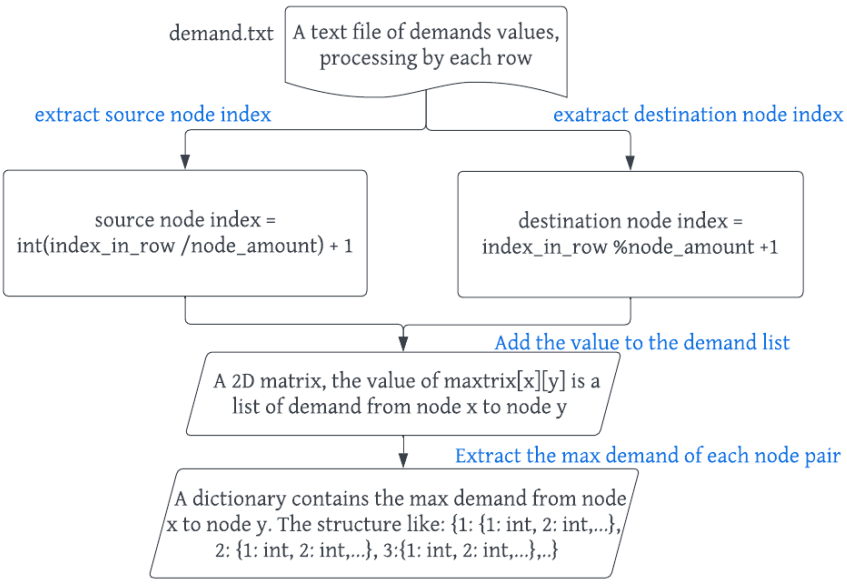

Figure 2 Parse the traffic demand.

2. 2 Algorithm implementation

Maximize the overall throughput of the network:

In traffic engineering design, our objective is to identify a network that can handle the maximum data flow within its capacity limits without risking link failure or causing surplus demand. To accomplish this, we can use Gurobi to design and solve the problem. The variables we will use are represented by f1, f2, ..., and fn, which denote the flow on the paths we select within the topology. By optimizing these variables, we can determine the most efficient network configuration that meets our data transfer requirements without overloading the network or creating vulnerabilities that could result in link failures. The next step is implementing the Objective. According to the goal of traffic engineering, which is to find a network that can carry the most data flow within the network so our objective is:

```
#create variable
flow = m.addVars(path,name = "flow")

#set objective
m.setObjective(flow.sum('*'),GRB.MAXIMIZE)
```

And then, we need to implement the constraints. The first constraint is the flow in the network should not exceed the capacity of each link. In this case, the first constraint is:

```
#add constraint: sum of flow on each edge <= capcity
for edge_key,edge_value in edges.items():
    m.addConstr( sum(flow[p] for p in edge_value) <= capacity[edge_key],"cap")
```

The second constraint is the flow on the network should not exceed the demand.

```
#add constraint: flow[p] <= demand
m.addConstrs((flow[p]<= demands[p] for p in path),"dem")
```

Minimize MLU (Maximum links utilization):

After maximizing the network throughput, we aim to increase the network's resilience to link failure. To achieve this, we implement a method that minimizes the maximum link utilization (MLU). MLU is the percentage of a network link's total capacity that is utilized by the traffic flowing through it. This value indicates the link's efficiency and can be calculated by dividing the actual data transmitted over the link by its maximum capacity. By minimizing MLU, we can distribute the flow more evenly across the network, which will increase its resilience to link failure. The Minimize MLU model builds upon the output of the Maximize overall throughput model. It is similar to the first model but includes an additional objective: minimizing the MLU (represented by the variable z). To calculate z, we need to implement a constraint such that the sum of the flow on each link is less than or equal to z. By solving this model, we can find the most efficient network configuration that maximizes throughput while minimizing the risk of link failure.

3. Results

Based on the first graph we generated, we can tell that there are 8 edges containing 1200 data flow. On the other hand, 7 edges carry 1200 data flow of the second graph, which means that the data flow on the network is more sparse than the first one. According to this evidence, we conclude that we can send as much traffic as possible by using the max throughput algorithm. However, we still need to consider the network's resilience. Hence, we need to compromise the maximum performance of the network, and minimizing MLU is one of the alternative ways to address this problem. Although the total throughput of Minimizing MLU is less than the throughput of the algorithm, we used to maximize throughput and did not sacrifice too much of the throughput while balancing the data flow on each edge of the network. This mechanism is a better solution to a typical network configuration since it provides more resilience and capacity for each edge to process unpredictable data flows. Thus, we can avoid undesired system crashes and maintenance.

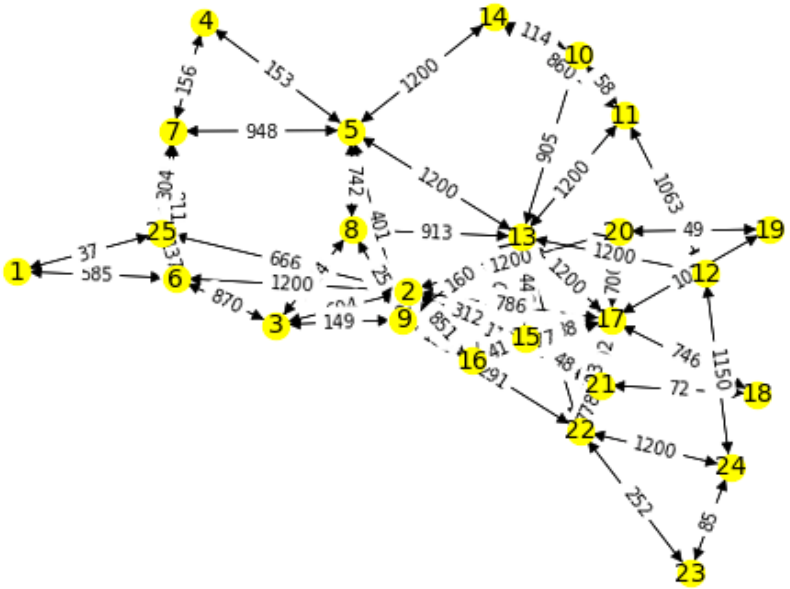

Figure 3 Result of max throughput.

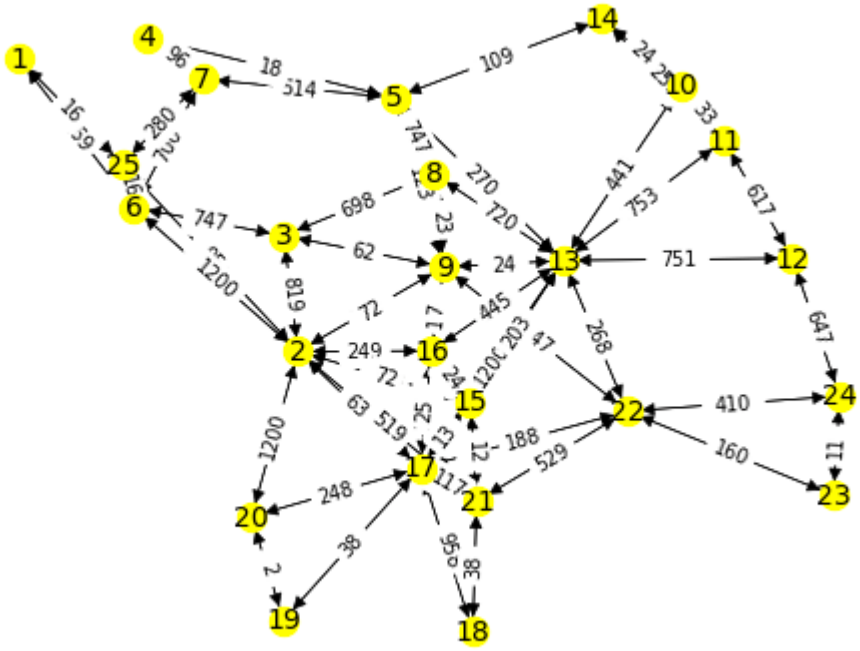

Figure 4 Result of min MLU

Future works:

Moving forward, we will select only one path from the source to the destination in our network configuration. This approach will allow us to utilize the k shortest path algorithm in future research to select multiple paths and provide greater path diversity in traffic engineering. By using more paths in the topology, the traffic engineering algorithm can distribute the flow more evenly across the network, which will increase network efficiency and resilience. Through this approach, we can explore a wider range of solutions to optimize the network's performance while reducing the risk of link failure.

6. Reference

Gurobi. (2022, August 24). Netflow.py example. Gurobi Optimization https://www.gurobi.com/documentation/9.5/quickstart_windows/cs_netflow_py_example.html

Manyaghobadi, T. (n.d.). Teavar/code/data at master · Manyaghobadi/Teavar. GitHub. https://github.com/manyaghobadi/teavar/tree/master/code/data

NetworkX. (n.d.). Tutorial#. Tutorial - NetworkX 3.1 documentation. https://networkx.org/documentation/stable/tutorial.html

Rachee Singh. (n.d.). Traffic engineering: From ISP to cloud-wide area networks. https://www.racheesingh.com/papers/te-survey.pdf

Racheesingh. (n.d.). Racheesingh/Cloud-te-tutorial. GitHub. https://github.com/racheesingh/cloud-te-tutorial

Wikipedia. (2023, March 30). Multi-commodity flow problem. Wikipedia. https://en.wikipedia.org/wiki/Multi-commodity_flow_problem