

Introduction

Efficient and scalable network design is a fundamental requirement for modern data centers, which contains diverse network traffic patterns. Datacenter networks must support such communication across any two servers within the datacenter, sometimes with immense volumes of data traffic, whilst taking maintainability into account too.

We examine Clos networks, originally conceived by Charles Clos in the 1950s to for telecommunications switching systems. Clos networks are composed of numerous small switches that group together to form a non-blocking network, allowing communication between any input-output pair without interference, even at high loads.

This is ideal in datacenters, where high bandwidth switches are exponentially expensive and technologically infeasible. By aggregating numerous affordable physical switches, we can aggregate the performance of a logically high performing switch.

Originally, Clos networks had three stages - ingress, middle, and output - due to the unidirectional nature of telecommunication. In datacenters, our Ethernet links are bidirectional. Thus, the popular approach is to fold the ingress and egress into one layer, replacing them instead by the edge layer of switches (where servers connect) and core layer of switches.

In this lab, we construct both a two-layer (folded Clos) and a three-layer (fat-tree) network programmatically; we aim to explore and quantify the impacts on network characteristics such as **redundancy**, **bisection bandwidth** and **switch count** by varying **switch degrees** as an input.

Implementation

Overall structure: Lebin implemented the overall structure of the Clos topology algorithm. It's a multi-layer network that composed by three different switch layers : edge, aggregation, and core. The edge switches connected to servers, aggregation switches in the middle that connected to the edge switches and the core switches at the top.

Algorithm input: The algorithm takes two inputs: the number of servers (n) and switch degree (d). The switch degree has to be even because each switch is split into downlinks (to servers or lower layers) and uplinks (to higher layers). If the switch degree is odd, the network can't be symmetric, so we enforced that the switch degree must be even.

Number of edges: The number of edges, aggregation and core switches are determined by the given switch degree. By the original fat-tree paper (three-layer Clos network), each switch is split evenly into downlinks and uplinks. Therefore, the formula I used for edge and aggregation was $\text{edge} = \text{aggregation} = d/2$ and core switch = $(d/2)^2$ to make sure full bisection bandwidth and full connectivity.

Number of Pods: Next, I created a pod class. Each pod in a Clos topology contains edge and aggregation switches. The number of pods depends on the number of servers and the switch degree. The formula I used was $\text{pods} = \text{ceil}(n / ((d/2)^2))$. By having the pod class, if two servers are in the same pod, traffic flows through the pod's aggregation switches (no need for core switches). Andrew implemented the two-layer

network, where there is no need for pods. Instead, each edge switch connects directly to a core switch, meaning that each edge switch is in a pod of its own.

Representing connections: We used adjacency lists with Connection objects and it will track the source and destination IDs from servers and switches. This made it easier to traverse the network for path analysis.

Analysis

Switch Count and Max Supported Servers

Given a switch degree d , we can analytically derive the maximum number of servers supported and the number of switches required to support them:

For the two-layer topology:

- Each edge has $d/2$ downlink, thereby supporting $d/2$ servers per edge. The $d/2$ uplinks each go to one core switch, so there would be $d/2$ core switches. Each core switch then has d links available, and each link goes to a edge switch, meaning there are d edge switches.
- Therefore, there are d edge switches, and $d/2$ core switches, supporting a maximum of $d^2/2$ servers.

For the three-layer topology:

- Each edge has $d/2$ downlink for $d/2$ servers per edge. Each uplink goes to an *aggregation* switch, so there are $d/2$ aggregation switches. Each aggregation switch will also have half its links reserved for edges, so there are $d/2$ edge and $d/2$ aggregation switches; these switches (and their servers) form a *pod*.
- Each uplink of an aggregation switch will need to connect to another pod's aggregation switch's uplink through a core switch. Since a core switch will also have d links, it means that each core will connect together d pods. Each pod has $d/2$ aggregation switches and each switch has $d/2$ uplinks for a total of $(d/2)^2$ uplinks per pod. Therefore, there needs to be $(d/2)^2$ core switches to aggregate each uplink per pod with all other pods.
- In total, there are $d/2$ edge switches and $d/2$ aggregation switches per pod, and $(d/2)^2$ core switches total. This gives $d^2 + (d/2)^2 = 5d^2/4$ total switches.
- Then, for d pods, each pod has $d/2$ edge switches with $d/2$ servers each, leading to $d * d/2 * d/2 = d^3/4$ total servers.

To test our implementation, we created topologies with $d=2, 4, 8, 16$ and the corresponding maximum number of servers supported. In *SwitchAnalyzer*, we simply count the number of switches our topology has, with results in the table below:

Maximum supported servers with $d=$	2	4	8	16	d
Two-Layer	2	8	32	128	$d^2/2$
Three-Layer	2	16	128	1024	$d^3/4$
Total switches needed with $d=$	2	4	8	16	d
Two-Layer	3	6	12	24	$3d/2$

Total switches needed with $d=$	2	4	8	16	d
Three-Layer	5	20	80	320	$(5d^2)/4$

From these, we see that our topology constructor correctly constructs the required switches, matching the expected theoretical value.

Redundant Links

A key characteristic of Clos topologies are its abilities to offer redundant links to communication between different servers. We wrote the `PathAnalyzer` (Lebin wrote the base for three-layer topologies, Andrew extended it to two-layer topologies) class to identify all paths between two servers. The algorithm performs a simple graph search to find all unique paths. We perform the search on our generate two- and three-layered Clos topologies with varying switch degree d , with results recorded in the table below:

Simulated with different $d=$	2	4	8	16	Theoretical redundancy with d
Two-Layer	1	2	4	8	$d/2$
Three-Layer	1	4	16	64	$(d/2)^2$

Our simulated topology outputs the expected count of redundant links. We provide analysis for why this is the case: the redundant links reside between the numerous connections between edge, aggregation, and core switches.

For two-layered topologies, it is relatively straightforward: each edge switch has $d/2$ uplinks to core switches, so paths that cross edge switches can take any of the core switches, leading to $d/2$ redundant links maximum.

For three-layered topologies, each edge switch has $d/2$ uplinks to aggregation switches. Then, each aggregation server connects to all core switches in a core group, which is another $d/2$ uplinks, for a total of $(d/2)^2$ unique paths.

Note that the table covers only the maximum redundant links. There are also connections which have only one path, namely paths between servers that share the same edge switch. These paths have only one path available, and are a source of potential failure. In three-layer topologies, servers within the same pod do not need to use core switches, these paths would only have $d/2$ redundant paths.

Bisection Bandwidth

Another desirable property that Clos networks have is that they have full bisection bandwidth. This means that for n servers that can each communicate at rate r , any half of the server can communicate with any other half at full rate, meaning that the bandwidth required is $(n/2)*r$.

For our Clos topologies, we can analytically solve for the desired bisection bandwidth:

- Given switch degree d , we have $2*(d/2)^2 = d^2/2 = n$ servers. Partitioning in half, we have $n/2 = d^2/4$ servers on each half, meaning we must support $r*d^2/4$ bisection bandwidth.
- $d^2/4$ servers share $(d^2/4)/(d/2) = d/2$ edge switches. Each edge switch has $d/2$ uplinks to core switches of bandwidth r . This means at the edge level, the bisection bandwidth is $d/2 * d/2 * r = r*d^2/4$, as desired.

For three-layered topologies:

- We must take into account pods. There are d pods, each containing $d/2$ edge switches and $(d/2)$ aggregation switches. The number of servers per pod is $(d/2)*(d/2) = d^2/4$, so the total number of servers across all pods is $d*(d^2/4) = d^3/4 = n$. Partitioning the topology gives $n/2 = d^3/8$ servers. Thus, the required bisection bandwidth is $r*(d^3/8)$.
- We first consider cutting at aggregation-to-core links: There are $d/2$ pods, each pod has $d/2$ aggregation switches, each with $d/2$ uplinks to core switches. Therefore, the number of aggregation-to-core links crossing the partition boundary is $(d/2)^3 = d^3/8$. Since each link has bandwidth r , the total bisection bandwidth at this layer is exactly $r*(d^3/8)$.
- Similarly, consider cutting at edge-to-aggregation links: There are $d/2$ pods, each pod has $d/2$ edge switches, each with $d/2$ uplinks to aggregation switches. Resulting again in $(d/2)^3 = d^3/8$ total crossing links. Each link has bandwidth r , yielding the desired bisection bandwidth $r*(d^3/8)$.

We implement BisectionAnalyzer, which partitions the servers "left to right" (servers with ID $0 \sim n/2$ are in one group). Then, we sum the bandwidths of the aggregation (if exists) and core layer links to arrive at the bisection bandwidth. Clos topologies are designed such that pods, switches, and links are uniformly distributed, ensuring identical connection patterns across the topology. Therefore, this partition naturally captures the minimal bisection bandwidth, and any asymmetric partition would increase the bandwidth. We provide the bisection bandwidth calculated by our analyzer in the table below:

Bisection bandwidth ($r=1$) for $d=$	2	4	8	16	d
Two-Layer	1	4	16	64	$d^2/4$
Three-Layer	1	8	64	512	$d^3/8$

As can be seen, our constructed topology correctly returns the desired full bisection bandwidth, enabling non-blocking communication across servers for any traffic.

References and Credits

The majority of the Clos topology constructor (3 layered) and the implementation section was written by Lebin.

The introduction and analysis of the writeup, as well as the testing classes and two layer Clos topology construction were written by Andrew.

We referenced the following sources:

- <https://packetpushers.net/blog/demystifying-dcn-topologies-clos-fat-trees-part1/>
- <https://packetpushers.net/blog/demystifying-dcn-topologies-clos-fat-trees-part2/>
- <https://textbook.cs168.io/datacenter/topology.html>