Simulation of a Scalable Commodity Data Center with Fat-Tree and Exploration of Future Alternatives.

Yilei Fu, Lihang Gong, Yuanfeng Wen

University of California, Berkeley

ABSTRACT

Today's data centers may contain tens of thousands of computers with significant aggregate bandwidth requirements. The network architecture typically consists of a tree of routing and switching elements with progressively more specialized and expensive equipment moving up the network hierarchy. [1]

In out team work, we simulated a special data center topology: Fat-Tree and applied its routing algorithm. Also, we applied flow classification and flow rearrangement on this data center topology.

Then, we found the Fat-Tree structure also have some drawbacks, which impelled us to find more new structures and new algorithms. We raise an overview of several future perspectives.

PURPOSE

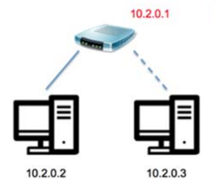
Our main purpose is to simulate the Fat-Tree data center structure and its routing algorithm. As we mentioned in abstract, we used Python and followed steps mentioned in the paper "A Scalable, Commodity Data Center Network Architecture". First, we applied addressing algorithm in order to build a basic Fat-Tree data center architecture, named every end host, switch and core switch and allocated their IP addresses. Then, we established a prefix/suffix storing system called "CAM" and "TCAM" structure. Although we cannot simulate the "Accessing Memory" part using Python, we still applied the routing algorithm -- we added prefix tables and suffix tables in order to run query on them when packets are routing, which is, 2-level-table lookup routing algorithm. 2-level-table lookup routing algorithm can help switches find their output ports effectively. When combining with TCAM architecture, the insert complexity can be O(N), and the delete complexity can be O(N^2). N is the length of prefix/suffix. Particularly, we simulated a whole routing path from 10.0.2.3 to 10.0.1.2. Details are mentioned in the "Simulation" part.

In addition, we applied the flow classification algorithm in order to recognize subsequent packets of the same flow, and forward them on the same outgoing port. Also, we are able to periodically reassign a minimal number of flow output ports to minimize any disparity between the aggregate flow capacity of different ports.

Nevertheless, Fat-Tree architecture and 2-level-table lookup routing example are not the only solution for data centers. We also introduced other structures such as VL2, and BCube.[2][3]

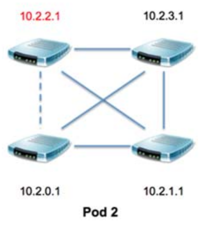
ROUTING ALGORITHM

We perform the following simulation of a packet from source IP 10.2.0.3 to destination IP 10.0.1.2:



Suffix	Output port
0.0.0.2/8	2
0.0.0.3/8	3

The routing table is stored at 10.2.0.1. Traffic goes evenly into upper-level switches, since the inter-pod traffic is switched.



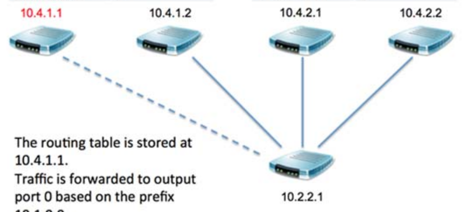
Prefix	Output port
10.2.0.0/24	0
10.2.1.0/24	1
0.0.0.0/24	

Suffix	Output port
0.0.0.2/8	2
0.0.0.3/8	3

The routing table is stored at 10.2.2.1. Traffic is forwarded to output port 2 based on the suffix 0.0.0.2.

Prefix	Output port
10.0.0.0/16	0
10.1.0.0/16	1

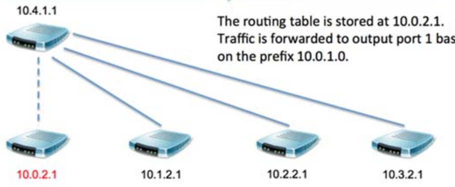
Prefix (cont'd)	Output port
10.2.0.0/16	2
10.3.0.0/16	3



The routing table is stored at 10.4.1.1. Traffic is forwarded to output port 0 based on the prefix 10.1.0.0.

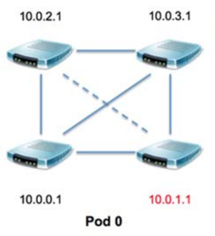
Prefix	Output port
10.0.0.0/24	0
10.0.1.0/24	1
0.0.0.0/24	

Suffix	Output port
0.0.0.2/24	2
0.0.0.3/24	3



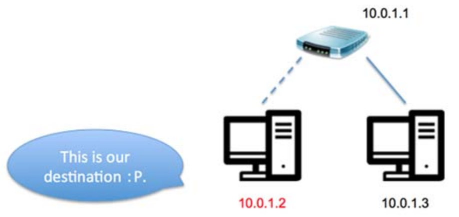
The routing table is stored at 10.0.2.1. Traffic is forwarded to output port 1 based on the prefix 10.0.1.0.

ROUTING ALGORITHM (CONT'D)



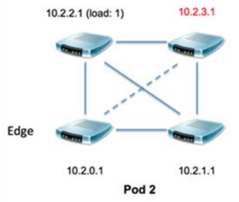
Suffix	Output port
0.0.0.2/8	2
0.0.0.3/8	3

The routing table is stored at 10.0.1.1. Traffic is forwarded to output port 2 based on suffix 0.0.0.2.



FLOW SCHEDULING

Suppose we have a new flow coming in at this point, which goes from source IP 10.2.0.3 to 10.0.0.2 (passing through the same pod switches):



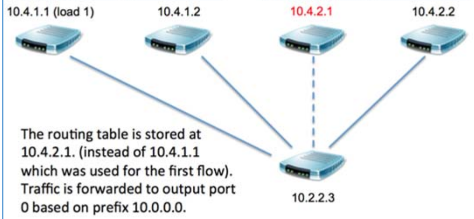
Prefix	Output port
10.2.0.0/24	0
10.2.1.0/24	1
0.0.0.0/24	

Suffix	Output port
0.0.0.2/8	2
0.0.0.3/8	3

The routing table is stored at 10.2.3.1. (Traffic needs to go evenly into upper-level switches).
Traffic is forwarded to output port 2 based on suffix 0.0.0.2.

Prefix	Output port
10.0.0.0/16	0
10.1.0.0/16	1

Prefix (cont'd)	Output port
10.2.0.0/16	2
10.3.0.0/16	3



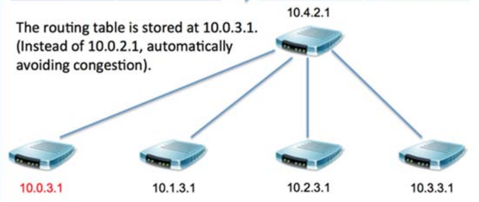
The routing table is stored at 10.4.2.1. (instead of 10.4.1.1 which was used for the first flow). Traffic is forwarded to output port 0 based on prefix 10.0.0.0.

FLOW SCHEDULING (CONT'D)

Prefix	Output port
10.0.0.0/24	0
10.0.1.0/24	1
0.0.0.0/24	

Suffix	Output port
0.0.0.2/24	2
0.0.0.3/24	3

The routing table is stored at 10.0.3.1. (Instead of 10.0.2.1, automatically avoiding congestion).



References

[1] Al-Fares M et al (2008), A Scalable, Commodity Data Center Network Architecture

[2]https://www.cs.cornell.edu/courses/cs5413/2014fa/lectures/08-fattree.pdf

[3]https://www.microsoft.com/en-us/research/publication/bcube-a-high-performance-server-centric-network-architecture-for-modular-data-centers/

You can find our full version of work on GitHub:
http://github.com/Fu-Yilei/EE122Project
E-mail: fuyilei96@berkeley.edu