# Asymptotically Stable Greedy Algorithm for Multi-path Load Balancing in Distributed Systems with a $k$-ary Fat Tree Structure

**Problem 1:** Datacenter Network Multi-path Load Balancing

**Team:** Bella y Sensual
**Coach:** Marcelo Lemus
**Authors:** Martín Andrighetti
Ignacio Muñoz
Vicente Opazo
Benjamín Rubio

# Asymptotically Stable Greedy Algorithm for Multi-path Load Balancing in Distributed Systems with a $k$-ary Fat Tree Structure

## 1 Abstract

In this article, we design a distributed system algorithm to transfer messages from access switches to servers in a $k$-ary fat tree structure, considering restrictions on message transmission, reception, and storage. The algorithm demonstrates strong performance across various test cases, achieving over 99% accuracy in message transfer and low latency.

**Keywords:** Distributed systems, Scheduling, Online Algorithms, Information Transfer Structures, Network Algorithms

## 2 Introduction

In the context of distributed systems, the efficient utilization of network resources is extremely relevant for optimizing performance on real world applications. This problem delves into multi-path load balancing within distributed systems, focusing on a $k$-ary fat tree structure. The goal is to design a robust algorithm that ensures the seamless transfer of messages while considering bandwidth limitations, buffer sizes, and network topology.

In this work we describe the approach implemented by our team in the competition, which consists of an heuristic algorithm that by leveraging the unique properties of a $k$-ary fat tree, aims to maximize success rates, minimize latency, and ultimately enhance the overall efficiency of the distributed system.

The subsequent sections will elaborate on the problem statement, underlying assumptions, and the intricacies of the proposed solution. Additionally, results from testing and optimizations will be presented, showing the algorithm's effectiveness in real-world scenarios.

This work contributes to the ongoing discourse in distributed systems, offering insights and solutions that could pave the way for enhanced performance and scalability in large-scale computing environments.

## 3 Problem Description

We start by a description of the problem. We work on a network which is guaranteed to be a $k$-ary fat tree, this is a common structure used in communication systems [1] that can be described as a graph with nodes of 4 types:

- Level 0 (Servers): Each server is connected to exactly one access switch.
- Level 1 (Access switches): Each access switch is connected to several servers and aggregate switches.
- Level 2 (Aggregate switches): Each aggregate switch is connected to several access switches and core switches.
- Level 3 (Core switches): Each core switch is connected to several aggregate switches.

Formally, the structure satisfies following properties:

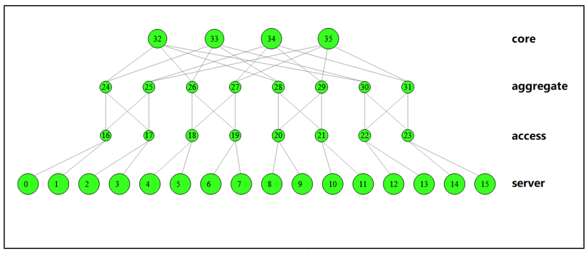

**Figure 1:** 4-ary fat tree example

1. The number of core switches is $(k/2)^2$.
2. It has k pods, a pod is a block which have $k/2$ access switches and k2 aggregate switches with all its access switches connected with all its aggregate switches.
3. There is $k/2$ core switches connected with the first aggregate switch of each pod, $k/2$ core switches connected with the second aggregate switch of each pod and so on.
4. Each access switch is connected to $k/2$ servers.

There is also an additional special controller node, but for the purposes of this article, we can ignore it as it was not used by our solution. As an example, Figure 1 shows the structure of a 4-ary fat tree.

In this problem we are required to send messages between the nodes of the network, for this purpose, each switch has three attributes:

- Bandwidth in: The maximum number of messages that the switch can receive at any time slice
- Bandwidth out: The maximum number of messages that the switch can send at any time slice
- Buffer size: The maximum number of messages that the switch can store at any time slice

The problem is divided into $T$ time slices. On each time slice a new request could appear in any access switch. The request has a target server and is divided into indivisible messages. If the correspondent access switch does not have enough buffer size to store the message it is lost. If all the messages reach the target server, the request is transmitted correctly.

We are required to construct a distributed algorithm that based only on information about the node states and limited information about the states its neighbors, decides on-line to which neighbor to send each of the messages on its buffer, restricted to not surpassing its bandwidth. If the message could not be received by the destination node, it does not leave the sender nodes buffer. A relevant restriction is that a message can not be sent to a server which is not its final destination.

At the end of each time slice each node constructs an information package to send to all of its neighbors and this package will be received by the neighbors at the start of the next time slice. This can be used to make more informed decisions on the next iteration.

For each completed request its latency is defined as the difference between the time when the request was completed and the time when the request was created. The success rate is the ratio between completed requests and the total requests.

The objective of the problem is to maximize the *Score*, which is calculated with the next formula.

$$SuccessRate*100 - \frac{Latency_{P99}}{20} - \frac{Latency_{avg}}{20}$$

Where $Latency_{AVG}$ is the average latency of the completed requests and the $Latency_{P99}$ is the 99th percentile of the latencies of all completed requests.

To maximize the score we need to create a strategy for a node (which messages and what information to send to the neighbors) without knowing the actions of the other nodes for this time slice.

## 4 Assumptions

Based on observations made from the testing data, we made the following assumptions regarding the parameters of the problem:

- The inbound bandwidth is significantly greater than the outbound bandwidth for all nodes.
- Buffer size increases with each level in the network, and the buffer size of the Core Switches is notably large.

## 5 Solution

With a clear understanding of the problem, we now turn our attention to the design of our solution. Our approach is guided by the objective of maximizing the number of successfully delivered requests while minimizing latency and ensuring efficient resource utilization across the network.

Firstly, we note that in order to maximize the number of successfully delivered requests, the algorithm should ensure that there is always free buffer space available at the access switches to accommodate any incoming requests and avoid rejections.

For this reason, the algorithm will attempt to maximize the utilization of the access switches' outbound bandwidth to keep their buffer empty. Conversely, it will utilize the aggregate and core switches' buffers as storage space to retain messages until the network has enough resources to transmit them.

Following this logic, we propose a distributed algorithm that manages message transfer and routing decisions at various levels of the network hierarchy. The following subsections delineate the protocols implemented at the access switches, aggregate switches, and core switches, each playing a different role in the network.

### 5.1 Access Switch Protocol

During any given time slice, after receiving new requests, the access switch buffer will contain messages destined both upwards and downwards. The algorithm will prioritize sending every packet going upwards until the outbound bandwidth is fully utilized. If there are insufficient messages going upwards to fill the bandwidth, the remaining bandwidth will be utilized to deliver messages going downwards to the servers.

To ensure that its bandwidth is consistently utilized to its fullest capacity, each access switch will reserve a small portion of its buffer space to store messages going downwards. To achieve this, whenever the switch is left with fewer messages going downwards than the defined reserved amount, it will transmit information to its parent aggregate switches requesting additional messages. Subsequently, during the next time slice, the aggregate switch will receive the information and transfer some of the messages stored in its buffer to the access switch requesting more.

Considering the latency of requesting new messages from the aggregates will be two time slices between the request and the reception of new messages, the reserved amount should be at least two times the outgoing bandwidth of the access switch. In this case, the worst-case scenario, in which there are no messages going upwards, is covered.

## 5.2 Aggregate Switch Protocol

The primary function of aggregate switches is to receive messages from lower switches and immediately redirect them upwards to the core switches. To achieve this, they allocate a small portion of their buffer exclusively for incoming messages, ensuring its constant availability. The remainder of the buffer is utilized to store messages from requests traveling downwards.

Upon receiving requests from lower nodes requiring additional messages, aggregate switches will dispatch a portion of the stored messages to the access switches. Similar to the behavior of access switches, if the quantity of downward-bound packets falls below the defined threshold, aggregate switches will request additional messages from the parent core switches to replenish their buffer.

## 5.3 Core Switch Protocol

Due to the substantial size of the core switches' buffers, their primary function is message storage. They utilize their outbound bandwidth solely in response to replenishment requests from their child aggregates, promptly dispatching additional messages as needed.

## 5.4 Considerations

### 5.4.1 Latency Reduction

To minimize latency across the entire solution, messages are sorted within each switch based on the start time of the request. This prioritizes older requests, ensuring their prompt transmission before newer ones. This sorting mechanism has shown significant latency reduction in the test data.

### 5.4.2 Message Distribution

Access switches determine which aggregate switch to send their messages to based on the smallest bandwidth capabilities of each aggregate in the path to the destination. Messages are distributed in proportion to these capabilities. For instance, aggregate switches with lower bandwidth receive fewer messages compared to those with higher bandwidth. This approach prevents bottlenecks and ensures equitable resource allocation within the network, leading to substantial performance improvements in final test data. It is hypothesized that this distribution strategy is most effective when there are significant variations in network capabilities across switches.

### 5.4.3 Parameter Optimization

To determine optimal values for parameters such as the size of reserved space on access and aggregate switches, we experimented with several different values and retained those yielding the best performance. After identifying the most effective parameter values, we achieved even better results on the test data.

## 6 Results

In the following section, we describe results obtained in the testing suite provided for the competition. Specifically, we look at the results obtained by our described approach and compare them to the final results obtained after the parameter optimization.

| Test | Ratio | L-AVG | L-P99 | Score |
|---|---|---|---|---|
| 1 | 0.99 | 13.52 | 52 | 95.74 |
| 2 | 1.00 | 5.52 | 9 | 99.27 |
| 3 | 0.99 | 8.39 | 19 | 98.49 |
| 4 | 1.00 | 6.43 | 21 | 98.62 |

Table 1: Results before parameter optimizations

| Test | Ratio | L-AVG | L-P99 | Score |
|---|---|---|---|---|
| 1 | 0.99 | 9.96 | 27 | 97.27 |
| 2 | 1.00 | 5.16 | 7 | 99.39 |
| 3 | 1.00 | 6.09 | 11 | 99.14 |
| 4 | 1.00 | 5.55 | 10 | 99.22 |

Table 2: Results after parameter optimizations

Our solution achieves commendable results, as evidenced by the tables displaying performance metrics before and after parameter optimization. In the initial tests, our algorithm demonstrated a high success rate, with over 99% of requests successfully transmitted.

However, there was room for improvement in latency, as indicated by the average latency (L-AVG) and the 99th percentile of latency (L-P99). After parameter optimization, significant enhancements were observed across all tests. The success rate remained consistently high, while both average and maximum latencies decreased noticeably. These improvements reflect the effectiveness of our approach in optimizing network performance and enhancing the overall efficiency of message transmission in distributed systems.

## 7 References

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A Scalable, Commodity Data Center Network Architecture.