**Day 11📑 Pseudo-code for Resource Allocation Algorithm**

Algorithm: Game-Theoretic Resource Allocation with Dynamic Pricing

Input:

U = set of users

F = set of fog nodes

For each user u in U:

demand\_u (CPU, memory, bandwidth)

latency\_req\_u

willingness\_to\_pay\_u

For each fog node f in F:

capacity\_f

base\_price\_f

Output:

Allocation Matrix (which user → which fog node/cloud)

Final Prices of each fog node

Metrics (latency, utilization, fairness, profit)

Steps:

1. Initialize:

For each fog node f in F:

price\_f = base\_price\_f

available\_capacity\_f = capacity\_f

allocation\_matrix = empty

2. Generate user requests:

For each user u:

record demand\_u, latency\_req\_u, willingness\_to\_pay\_u

3. Repeat until all users are processed OR no resources left:

3.1 Select a user u (random order or priority-based).

3.2 For each fog node f in F:

if available\_capacity\_f >= demand\_u:

compute user utility:

utility\_u\_f = willingness\_to\_pay\_u - price\_f \* demand\_u - latency\_penalty

else:

utility\_u\_f = -∞ (cannot be served here)

3.3 Select fog node f\* with maximum utility\_u\_f

3.4 If utility\_u\_f\* > 0:

allocate demand\_u to fog node f\*

update available\_capacity\_f\* -= demand\_u

update allocation\_matrix[u] = f\*

Else:

send user to Cloud (high latency, fixed cost)

update allocation\_matrix[u] = Cloud

3.5 Update prices:

For each fog node f:

if demand > supply:

price\_f = price\_f + Δ (increase price)

else if demand < supply:

price\_f = price\_f - Δ (decrease price)

4. End Loop

5. Compute metrics:

- Resource utilization per fog node

- Average latency per user

- Fairness index

- Provider profit

6. Output:

allocation\_matrix

final prices of fog nodes

metrics

**📘 Example Walkthrough**

* **Input:** 3 users, 2 fog nodes.
  + User 1: demand=5, max pay=10
  + User 2: demand=8, max pay=6
  + User 3: demand=12, max pay=15
  + Fog 1 capacity=10, Fog 2 capacity=15, base price=$1/unit
* **Step 1:** Initialize prices = 1.
* **Step 2:** Process User 1 → can pay → gets Fog 1.
* **Step 3:** Process User 2 → utility too low → moved to Cloud.
* **Step 4:** Process User 3 → gets Fog 2.
* **Step 5:** Prices updated (Fog 1 full, price ↑; Fog 2 partly full).

👉 **Output:** Allocation matrix + new prices.

***IEEE Style***

The Main() routine, as shown in **Algorithm 1**, is responsible for collecting and processing user demands.

* The **first for loop** collects and stores the demands in the list list.
  + Each user is given an equal chance of being processed first by randomly distributing the demands in the list.
* The **second nested for loop** iterates through the list list one by one.
  + Within this loop, the **inner for loop** is used to check if the current user’s demand can be accommodated within the available resources for the duration of their deadline t\*.
    - If the user’s demand **cannot be met** based on the available resources, the rest() function is called, indicating that the user’s demand for this round will be denied.
    - On the other hand, the pricingCalculation() subroutine is used to **dynamically compute the user’s pricing** based on the prior demand patterns to see if the user’s need can be met.

