Analysis and Proof

Objective:

We want to prove that for a given start and end time of an interval, and a sequence of packets with arrival times, profits and deadlines, our algorithm maximizes

$$\sum \text{profits of packets}$$

that can fit inside this interval for a valid RU configuration.

Notations:

Let ω be the number of distinct modes one RU can take (eg: 484 tonnes, 242 tonnes, etc).
Let $m[\omega]$ be array of size ω where $m[i]$ denotes max occurence of RUs we can have of the i'th mode. Assume that we consider them in descending order of values ($m[i] >= m[j] \; when \; i < j$) such that RUs of mode j can be split into RUs of mode i if $j > i$.
Let any config be denoted as an array of size ω where for each $i = [1, \omega]$, config$[i]$ denotes the number of RUs available of i'th mode in that config.
Let pconfigs be a 2D array denoting all the possible valid configurations of RU distribution.

Algorithm:

Phase 1 (filling base config):

- Maintain ω min priority queues ($pq[\omega]$) each allowed to have size at max $m[i]$ corresponding to base config for $i = [1, \omega]$
- Process the packets in any order, for each packet iterate over RU configurations : $i = [1, \omega]$:
  - if packet cannot be transmitted in mode i for current interval, continue to $i + 1$
  - else if $pq[i]$ isn't full then insert this packet to $pq[i]$ and continue to process next packet
  - else if $pq[i]$ is full but the top packet in $pq[i]$ has less profit than current packet, then poll the packet in $pq[i]$ and insert current packet. Now continue this process for next i but with the polled packet
  - else continue to next i
- Now for each of the ω modes, we have best set of packets that would go into that config.

Phase 2 (selecting optimal valid config):

- Iterate over the configurations in pconfigs. For a given config:
  - Iterate over all the modes as $i = [1, \omega]$:
    - Select $\min(pq[i].size, config[i])$ best packets from $pq[i]$ and add their profits to current score.

- Now we have the score for this config, if this is the best score so far, select this config and the packets associated with it as optimal.
- We now have our optimal configuration and the set of packets that fit with that configuration.

Proof:

Let A be the set of optimal packets that can be assigned to this interval for any valid configuration.
$optimalProfit$ = $\sum profits\ of\ packets\ in\ A$

Claim 1:

For any packet in A, let it be assigned to RU of mode $i \in [1, \omega]$ in the optimal configuration, then i is the lowest mode of RU the packet can be transmitted in.

Proof: If there existed a smaller mode j where we can assign the packet to, then in the optimal configuration we split RU of mode i into mode j (since mode j is of lower bandwidth than mode i). By doing so all the packets in A will still be mapped to an RU to be transmitted by, and some additional RU's might be introduced by this process, leading to a score >= optimal score.

So each of the packets in A would be transmitted in the RU corresponding to its lowest mode of RU.
Let $optimalConfig$ be the optimal configuration that transmits all the packets in A.

Claim 2:

While calculating the score over a configuration in phase 2 of the algorithm :

$$Calculated\ score\ for\ optimalConfig = optimalProfit$$

Proof :
Let $assignedAt[i]$ denote the collection of packets in A that are transmitted using RU's whose mode is i where $i \in [1, \omega]$.
Also let $sz[i]$ denote the number of packets in $assignedAt[i]$.
It suffices for us to show that

$$\sum profit\ of\ top\ sz[i]\ packets\ in\ pq[i] = \sum packet\ profits\ in\ assignedAt[i]$$

$\forall\ i \in [1, \omega]$

Consider any packet p in $assignedAt[i]$. There are 3 cases:
Case 1: p lies somewhere in $pq[j]$ where $j < i$:
This case implies that p can be assigned in RU with mode j. But this contradicts our Claim 2, thus this case is not possible.
Case 2: p lies somewhere in $pq[j]$ where $j > i$:

This case arises only when there exists another packet $p2 \notin A$ whose profit is $>= p$ and can be transmitted in mode i. This is because p must have tried to fit inside $pq[i]$ but it was either polled from $pq[i]$ due to some $p2$ with greater profit or p could not have been inserted to $pq[i]$ due to having lesser profit than some $p2$ present at the top of $pq[i]$. Either way swapping $p2$ with p in A will gives us the optimal answer.

Case 3: p lies in $pq[i]$:

In this case, p would be part of the top $sz[i]$ packets of $pq[i]$ since if that were not the case then we'd either have other packets which do not belong to A and have profits $>= p$'s profit, which could replace p in A.

Note that $pq[i]$ would atleast have as many packets as $assignedAt[i]$ $\forall i \in [1, \omega]$ because we took the size of $pq[i]$ to be $m[i]$ which is the max occurence of RUs in mode i.

Therefore all the packets in A are assigned in their respective modes in $pq[\omega]$ priority queues and taking the summation for $optimalConfig$ would lead to $optimalProfit$.

Time Complexity analysis:

Let M = $\sum_{i=1}^{\omega} m[i]$
and β = number of possible configurations then
Then Phase 1 of the algorithm takes $O(n + M \log M)$ time complexity.
Phase 2 of the algorithm takes $O(\beta * M)$ time complexity.

Specific to our case we have $M = 33$ and $\beta = 36$.

Pseudo Code (needs refinement)

Best packets for an interval selector function

```
let num_modes <- number of possible modes of RU
let max_occ[num_modes] <- array containing max occurences of RUs in a mode in
sorted fashion
# example {18, 8, 4 ... , 1}

func getBestPackets(currentInterval, availablePackets):
        Priority Queue topPackets[num_modes]
        for pckt in availablePackets:
                For mode from lowest to highest:
                        if(pckt can fit into currInterval)
                                if(sizeof topPackets[mode] < max_occ[mode]):
                                        topPackets[mode].insert(pckt);
                                        move on to next pckt
                                else if(topPackets[mode].top.profit <
pckt.profit):
```

```
                                        insert pckt
                                        poll topPackets[mode] and assign that
to pckt
                                        contine process for the polled packet
(new pckt)

        bestconfig = null, bestscore = -1
        for config in configs:
                get score of config
                (by taking best config[mode] packets from topPackets[mode])
                if(bestscore < score):
                        swap(bestconfig, config)
                        bestscore = score

        for bestconfig get packets
        return packets, bestconfig
```

Main Function

```
selectedIntervals = {}

For d = [1,Delta]:
        For t = [0, TimePeriod-d]:
                currentInterval <- {t, t+d}
                mpackets, bestconfig <- getBestPackets(currentInterval,
availablePackets)
                drop mpackets from availablePackets
                insert mpackets to currentInterval
                curscore = summation(profit of packets in currentInterval)
                crossIntervals = set of intervals in selectedIntervals
                ixscore = summation(profit of packets in crossIntervals)
                if(ixscore*2 < curscore) :
                        drop crossIntervals from selectedIntervals
                        add all the packets in crossIntervals back to
availablePackets
                        insert currentInterval into selectedIntervals
                else :
                        add mpackets back to availablePackets
```