**Computer-Communication PS2**

**Daniela Shabat 316415645**

**Mor Berkman <ID>**

<https://github.com/danielashabat/computer-communication/tree/main/Ex2_WFQ>

Modules:

queue\_and\_package module contains the data structures of Package and Queue and related functions of those data structures.

Data structures:

|  |  |
| --- | --- |
| typedef struct Package { | |
|  | int time; // save the arrival real time of the package |
|  | char Sadd[20]; //source address |
|  | int Sport; // source port |
|  | char Dadd[20]; //destination address |
|  | int Dport; //destination port |
|  | int length; //packet’s length/size |
|  | float weight; |
|  | float last;// virtual last time |
|  | bool print\_weight; //if true print the weight in output report |
|  | bool ignore; // if true the package will be ignored |
|  | struct Package\* next; // next linked package |
|  | struct Package\* prev; // previous linked package |
|  |  |
|  | } Package; |

In this data structure we save all the data regarding to the specific package. Each packet is linked to more packages with the same flow (same source address, source port, destination address, destination port) with a link list strategy (for hence we have the prev and next field).

|  |
| --- |
| typedef struct Queue { |
|  | int size;//the amount of packages in queue |
|  | Package\* head;//pointer to the first package in queue |
|  | Package\* tail;//pointer to the last package in queue |
|  | char Sadd[20]; //source address |
|  | int Sport; // source port |
|  | char Dadd[20]; //destination address |
|  | int Dport; //destination port |
|  | float weight; |
|  | struct Queue\* next; //next linked queue |
|  | }QUEUE; |

Each Queue data structure has a distinctive flow. When a new package arrives with the same flow, the package inserted to the end of it matching queue. The weight field is updated when there is new package with a different weight of the matching queue. The default weight of queue is 1. The queue data structure work as well as linked list, where the queue is linked to other queues in the program.

The relative connection of the data structures:

Queue 0 : pack 0 -> pack 1

Queue 1: pack 0 ->pack1 ->pack2

Queue 2: pack 0

Each queue is linked to another queue and each queue contains a linked list of packages.

Important functions:

/\*this function search for the right queue for the new package and insert it to the queue, if no queue found it will create new queue and insert the package\*/

void InsertNewPackage(QUEUE\*\* ptr\_head, Package\* new\_package);

/\*this function returns the package with minimum last that waiting to be sent\*/

Package\* GetPackageWithMinimumLast(QUEUE\* head);

// this function returns the sum of all active links: the function searchs in each queue the first package with last that bigger than round\_t (that’s mean the link is active) and sum it up.

float SumActiveLinksWeights(QUEUE\* head, float round\_t);

// the function searchs for last==-1 in all packages, and update it. Explain: when we first insert package we insert it with a default last=-1 . after we calculate the round\_t we need to update the last with the according function.

void UpdateLast(QUEUE\* head, float round\_t);

complexity:

the complexity in each packet derived from the round(t) calculation. When we calculate round(t) we need to check if arrival or departure is the next event. If arrival is the next event it only takes 1 calculation, if departure is the next event we need to make 1 calculation plus repeating the same comparison until we get that arrival is next event.

In the worst case we can imagine there is N iteration of departures, it means O(n). in our tests we can see in average there is 1 departure in each arrival that’s mean O(1).