PA2 – WFQ

Submitted by: Nicole Frumkin (ID 211615372) & Or Shalev (ID 211381942)

Our code implements a Weighted Fair Queuing scheduler. It reads each line (that represents a packet) from stdin and outputs the transmission time of each packet to stdout. It prioritizes the packet by calculating the virtual time of each one and choosing the minimal one. In addition it supports weighted packets and handles simultaneous connections transmitting. Each connection is defined by the four parameters: source IP, destination IP, source port and destination port.

Structs

1. **Packet** – this struct is used to save the parameters read from each line:
   `Time, Sadd, Sport, Dadd, Dport, packetLength, [weight]`
   In the start of the program we initialize an array of packets in the length of a few thousand hundreds of packets, and each time we read a line we save the packet in the array.
   In addition to those parameters, we also store:
   - **`virtualFinishTime`** – calculated each time by the formula we saw in the recitation:
     $$last\big(p(i)\big) = \max\big\{round(t), last\big(p(i-1)\big)\big\} + \frac{size\big(p(i)\big)}{w_{flow}}$$
   - **`hasweight`** – this field is used to determine whether we need to update the connection’s weight or make the packet inhert the connection’s weight if it is different from zero. We also use it to determine whether to print the weight or not to stdout.
   - **`connectionID`** – it is used to store the connection corresponding to the packet.
2. **Connection** – this struct is used to save each packet in its corresponding connection. It is made of the four parameters that represent a connection: Sadd, Sport, Dadd and Dport, and addtionaly stores the:
   - **`weight`** – the weight of the connection. It changes if a packet has a weight and is not changed till the next time a packet changes its weight.
   - **`virtualFinishTime`** – in addition to the virtual finish time of each packet we store the virtual finish time of each connection to determine whether it is available to send data or not.
   - **`queue`** – this is a struct that is used to stores all the packets that are assigned to the connections. Once a packet is printed to the file it is removed from the queue.
3. **`Queue`** – a simple struct to store the packets items. It has a queue from the type `Packet*` and the regular parameters such as front pointer, rear pointer and queue size.

When the algorithm drains packets from the queues, it first checks whether there is a connection transmitting currently. If there is, it adds its weight to a variable called:

$$activeLinksWeight$$

We also use a global variable called globalFinishTime that is used to calculate when to schedule the next packet and transmit it. We use previous weights variable to insert it into the following formula:

$$round(t + x) = round(t) + \frac{x}{\sum_{activelinks} Weights\,|_t}$$

Functions:

1. **`int findOrCreateConnection(Packet *packet, int *connectionCount, Connection *connections, int packetCount)`**
   After saving the packet parameters, this function iterates over the current connections and searches for the corresponding connection for the packet. If it exists – it simply returns the idx of the connection. If it doesn't – it initializes a new connection and its queue.
2. **`void printPacketToFile(Packet *packet, int actualStartTime);`**
   This function simply prints the packet in the correct format.
3. **`void savePacketParameters(char *line, Packet *packet);`**
   It saves the packet parameters in the correct fields. If it has a weight specified, it also assigns that weight to its connection. Else, it inherits the connection's weight.
4. **`void drainPackets(Connection *connections, int connectionCount, int remaining);`**
   This is the main logic function.
   First it iterates over each connections summing the weights of the active links.
   Then It iterates over each connection again, this time looking for the packet with the minimal virtual finish time.
   If there is one, it prints it to stdout and removes it from its queue. It also updates the globalFinishTime.
   If there isn't one, it advanced globalFinishTime to the next packet arrival time.

Complexity Analysis

- Time complexity
  Let n be the number of packets and f be the number of flows.
  `findOrCreateConnection` runs in $O(f)$ because it linearly scans the connections.
  Each queue function (such as `dequeue`, `enqueue` or peek) runs in a fixed time $O(1)$
  The main loop scan all connection for each packet, meaning it runs $O(f)$ and because we have n packets in the worst case our algorithm runs in:
  $$O(n \cdot f)$$
- Space complexity
  We have 2 structs – one in size $O(n)$ for all the packets and one in size $O(f)$ for all the connections. So in total:
  $$O(n + f)$$