PA2_Computer_Communications

A project in Computer Communications.

Code Structure

The `PacketInfo` class represents information about a packet, as read from stdin: time, connection (source ip, source port, destination ip, destination port), length, and optional weight.
It has a `parse` method that constructs a `PacketInfo` from a string, and implements the `<<` operator, in order to print `PacketInfo`'s to stdout.

The `ChannelInfo` class contains information about a channel (that is, a set of packets with the same connection string). Each `ChannelInfo` has an index (the index of this channel among the channels seen in stdin); a weight (the current weight of the channel); and a queue of `PacketInfo`'s, which stores packets to send on this channel.

The global hash map `channelMap` maps connection strings to channels.

The global priority queue `active_channels` stores all the active channels (that is, channels that have at least one packet to send), ordered by their priority.

We created a helper class `ActiveChannelEntry` for the priority queue. `active_channels` is a priority queue of `ActiveChannelEntry`'s. Each `ActiveChannelEntry` contains a pointer to a channel, and the channel's weight at the time it was added to the priority queue. `ActiveChannelEntry` implements the comparison operators, so it can be stored in a priority queue.

The functions `read_batch_with_timeout`, `read_batch`, and `read_with_timeout` allow us to read packets from stdout in groups, instead of one at a time:

- `read_with_timeout` reads all the packets that arrived until some time limit (given as a parameter).
- `read_batch` reads a packet, and then also reads all the packets that arrived at the same time as that packet.
- `read_batch_with_timeout` combines the functionality of `read_with_timeout` and `read_batch`: it reads a group of packets that arrived at the same time, but with a time limit.

All of these functions add the packets they read to the appropriate channels, and also update the priority queue if necessary.

The main function works by the following logic:

1. Read the next batch of packets (a group of packets that arrived at the same time), and add them to the appropriate channels.
2. Get the best channel from the priority queue.
3. Pop the first packet from that channel's queue.
4. "Transmit it" (print it to stdout).
5. Check if new packets have arrived while transmitting this packet, and if so, add them to the appropriate channels.
6. Repeat until all packets have been transmitted.

Computational Complexity

For each packet we read, we have to:

1. Read it from stdin.
2. Look for its channel in a hash map, or maybe create a channel and add it to the hash map.
3. Add the packet to the channel.
4. Maybe add the channel to the priority queue.
5. Later, pop the channel from the priority queue.
6. Remove the packet from the channel.
7. Again, maybe add the channel to the priority queue.
8. Print the packet.

Most of these actions are `O(1)`. Updating the priority queue is `O(log n)`, where `n` is the number of active channels. Updating the hash map is `O(1)` amortized, but `O(m)` in the worst case, where `m` is the number of channels. Updating the channel is `O(1)` amortized, but `O(k)` in the worst case, where `k` is the number of packets on the channel.

So, the average complexity of processing each packet is `O(log n)`, where `n` is the number of active channels. The worst case is `O(log n + m + k)`, where `m` is the number of channels and `k` is the number of packets on the packet's channel.

The whole program's worst-case complexity is `O(N log N)`, where `N` is the total number of packets.