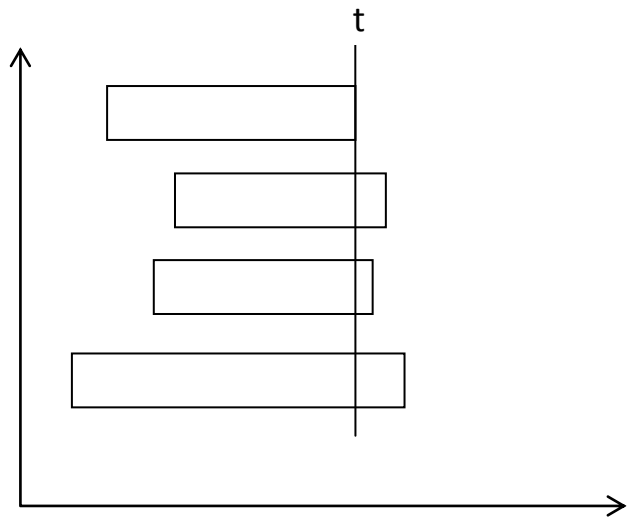
CS 7260 – Internet Architecture and Protocols

Lecture of 2-Nov-2012 **Raghavendra Pai**

This lecture distinguishes the standard WFQ implementation from a new one. I shall refer to the old algorithm as 'Standard' and the new one as 'New' (implementation).

There is usually a separate data structure (such as a heap) for finding the next packet with the earliest GPS finish time.

The standard implementation has the following problem. If a lot of packets have their GPS finish times at about the same real time (say, in order of a million), in that case it will be difficult to process all packets successively in close succession, in the limited interval. It is also not possible to precompute the real finish times (given the virtual finish times) of these packets, because a new first-in-session packet could invalidate the computation. However delaying would necessitate an infinitely fast processor – and hence the following problem is referred to as Mandatory Lazy Evaluation in [Zhao, Q., Xu, J. ``On the Computational Complexity of Maintaining GPS Clock in Packet Scheduling'', in Proc. of IEEE Infocom 2004.]



Consider the case when there are 1 million flows like these - computation of the real finish time is not practical.

The New algorithm in [P. Valente, “Exact GPS simulation with logarithmic complexity, and its application to an optimally fair scheduler“, IEEE/ACM Transactions on Networking (ToN), February 2008.] is explained below.

The algorithm tries to calculate the virtual time of the packet when a new packet arrives.

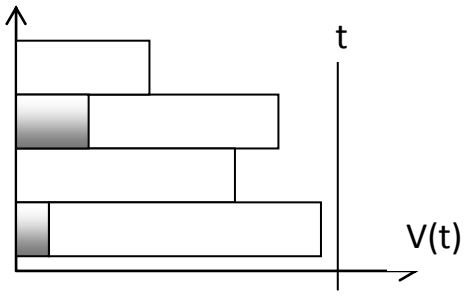
There are 2 ways this could be done:

Naive algorithm:

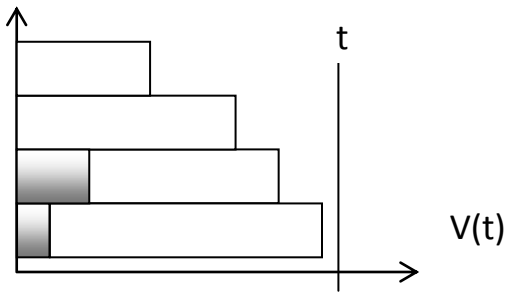
The relative location of the packet can be found by calculating the real time, starting from the last recorded t, between every 2 events (such as when a packet is processed) and then summing these up till the current time t. This is a O(N) algorithm. This is prohibitively large for practical applications.

New algorithm:

Instead of maintaining only the head-of-line packets only in the data structure, the New algorithm stores all the packets in each flow in the data structure (or rather their representation/pointer) so that a ‘contour’ is generated. The flows are then sorted by their total length. The resulting structure is called a ‘ladder’.



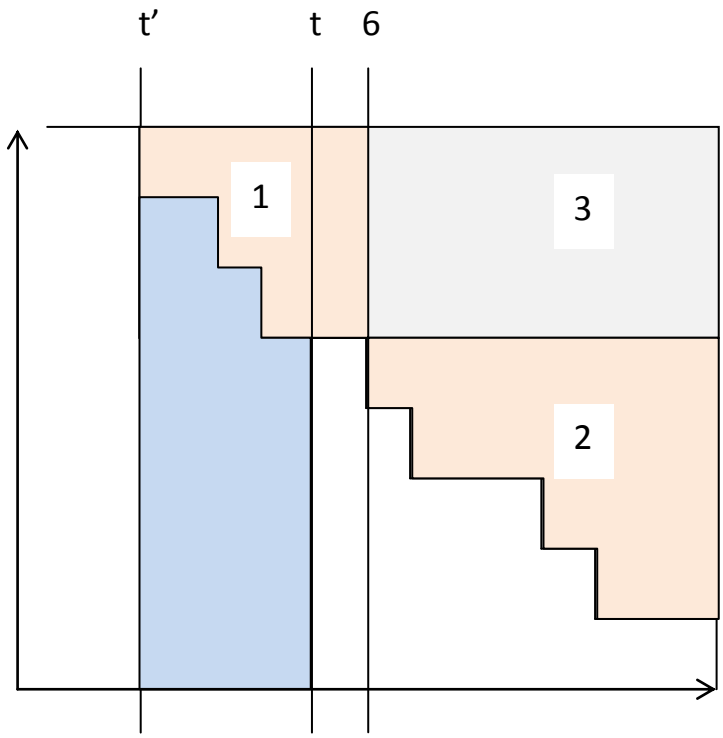
contour of flows



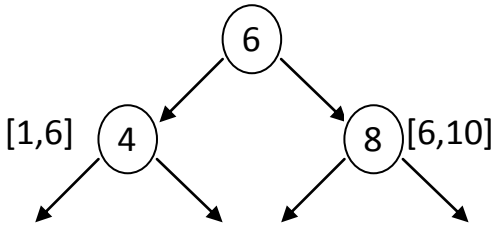
ladder of flows

Once the virtual time was found, packets could now be inserted/deleted/updated in the priority queue. In the Standard algorithm, the contour is a contour of HOL packets, while in the New algorithm, it comprises the last-in-session packets of the flows. The standard algorithm requires support for extract_min() and insert() and a heap is sufficient. But the New algorithm requires a balanced tree to enable a O(log(N)) binary search on the events.

The balanced tree data structure can be constructed dynamically as packets arrive, and searches are by events (arrivals or departures of flows)



From the diagram or the data structures, we can construct the binary search tree, rooted at 6, accounting for all events – each event represented by a node with fields for area above (or the area below) i.e. 6 i.e. [1,10] will store the entire upper area A=A1+A2+A3. 4 [1,6] will store the left area A1. 8 [6,10] will store the right area A3. The branching continues till all nodes are accounted for. Search takes O(log(N)) where N is the number of nodes.



Now the virtual time for the packet arriving at the current time t can be found from comparing with each node and taking the appropriate branch. Once the virtual time is known the new flow can be inserted into the balanced tree, which could be an AVL tree, which would then have to be rebalanced. There can be a separate heap for finding the packet with the earliest GPS finish time.

Details about the augmented data structure required for supporting the above operations and the corresponding invariant shall be covered in the next class.