**CMPSCI 754 Multimedia Systems** **Spring 2001**

Lecture 20: Apr 19

*Lecturer: Prashant Shenoy* *Scribe: Ping Ji*

# Network Packet Scheduling

## 20.1 Overview

Packet schedulers sit at the *output queues* of a switch. Packet scheduling algorithms determine which packets to drop, thus control the packet delay and packet loss for a flow. For real-time scheduling, process $i$ has model $(c_i, p_i, d_i)$, where $c_i$ represents CPU service; $p_i$ represents period of arrival, and $d_i$ denotes the deadline of $i$. Figure 20.1 shows the basis of real-time scheduling.

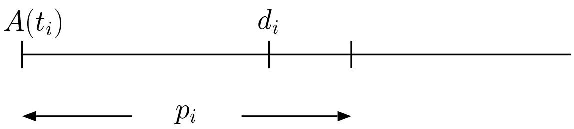


Figure 20.1: Basis of Real-time Scheduling Algorithm

$A(t_i)$ is the arriving time of $i$. The relationship between $p_i$ and $d_i$ is ignored. And $A(t_i) + d_i$ is used by CPU to schedule. The total service asked by tasks should not be greater than what CPU can provide, $\sum_{i=1}^{N} \frac{c_i}{p_i} \leq 1$.

To solve the problem that EDF scheduling does not protect flows from each other when sources do not conform to the specification, we assign a rate to each flow and compute deadline based on expected arrival time (EAT).

$$EAT_i(k) = EAT_i(k-1) + \frac{l_i}{r_i}$$

$$d_i = p_i = \frac{l_i}{r_i} \quad c_i = \frac{l_i}{C}$$

Schedulability constrain,

$$\sum_{j=1}^{N} \frac{c_j}{p_j} \Rightarrow \sum_{j=1}^{N} \frac{r_i}{C} \leq 1$$

Here, the assigned rate $r_i$ doesn't depend on the rate of sending packets.

A problem is arisen: *A source may send too slow and then very fast, leading to violations of QoS for other flows.* Figure 20.2 shows an example of this problem.

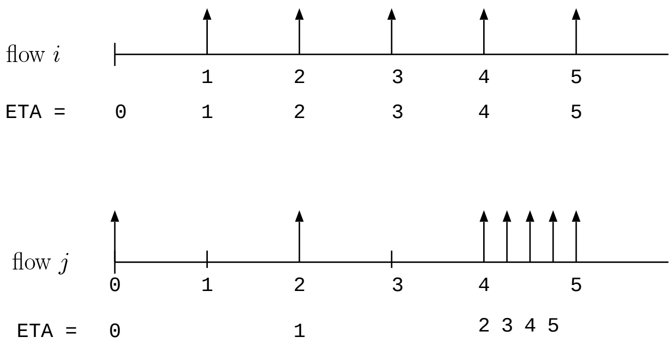


Figure 20.2: Problem of Real-time Scheduling Using EAT

Unfairness happens in this case.

## 20.2 Delay EDD, Jitter EDD and Virtual Clock Scheduling Algorithms

*Delay EDD* assigns a deadline to each packet. The packets are sent in order on increasing deadlines. The assigned deadline is just the expected arrival time (EAT) added to the delay bound at the server, $EAT_i(k) + d_i$. Delay EDD seperates rate and delay allocations.

*Jitter EDD* extends Delay-EDD to provide delay-jitter bounds (that is, a bound on the minimum as well on the maximum delay). A packet is hold until $EAT_i(k) = \max\{EAT_i(k-1) + \frac{l_i}{r_i}, AT_i(k)\}$. An example is shown in Figure 20.3. By Jitter-EDD, a flow sending slow at beginning will not get benifit by sending fast later, because of the way that Jitter-EDD computes the bound.

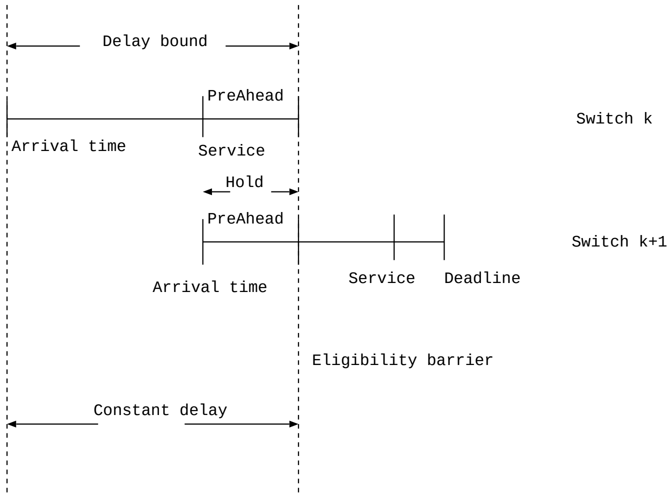


Figure 20.3: Packet Service in Jitter-EDD

A *comparison* of Delay-EDD and Jitter-EDD is shown in Figure 20.4,

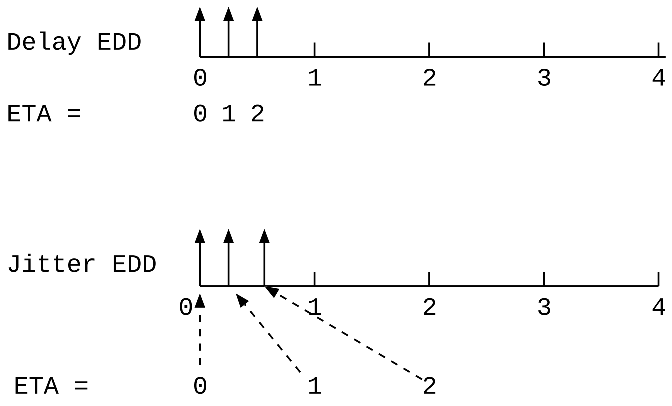


Figure 20.4: Comparison of Delay-EDD and Jitter-EDD

Using Jitter EDD, after the 1st packet is sent out, the 2nd packet is hold until it gets to the $ETA$. The out going flow also follow the incoming order. Thus a flow is smoothed.

*Virtual Clock* allocates a virtual transmission time to each packet, $VC_i(k) = EAT_i(k) + \frac{l_i^k}{r_i}$, which is the time at which the packet would have been transmitted were the server actually doing the Time division Multiplexing (TDM) [1]. Here, similar to Jitter-EDD, $EAT_i(k) = \max\{EAT_i(k-1) + \frac{l_i}{r_i^{k-1}}, AT_i(k)\}$. Packets is scheduled in the order of their VC values. If $\sum_{j=1}^{N} r_j \leq C$, then a packet is transmitted by a VC scheduler by $VC_i(k) + \frac{l^{max}}{C}$, where $l^{max}$ is the maximum packet size. A simplified example: if a client is to get a service rate of 5pkts/sec, incoming packets from that client are stamped with virtual service times 0.2 seconds apart.

We define that *work conserving* is that as soon as a packet arrives, the network serves it; on the other hand, *non-work conserving* is that when a packet arrives, the network can still be idle. Delay-EDD and Virtual Clock scheduling algorithms are work conserving, and Jitter-EDD is non-work conserving. In the meantime, Delay-EDD and virtual Clock allow sources to utilize idle bandwidth, but later penalize them (i.e., they are unfair).

## 20.3 Fair Algorithms

In fair packet scheduling algorithms, flows are assigned weights. Flows receive bandwidth in proportion to weights. The ideal case of fairness is to minimize the difference of the *bandwidth-weight ratio* between flows, $\frac{W_f(\Delta)}{r_f} - \frac{W_m(\Delta)}{r_m} = 0$, where $W_f(\Delta)$ denotes the service received by flow $f$ and $r_f$ denotes the weight of flow $f$. In general, the goal of fair algorithms is to minimize the unfairness, packet-by-packet service $|\frac{W_f(\Delta)}{r_f} - \frac{W_m(\Delta)}{r_m}| \leq H(f,m)$, where $H(f,m)$ measures the unfairness.

*Weighted Fair Queuing (WFQ)* schedules the flow in a round robin fashion, using the time of last packet finished to order the schedule time. A figure is shown in slide 150. A finish tag $F(p^j)$ is defined to represent the round in which packet finishes service, $F(p^j) = S(p^j) + \frac{l^j}{r}$ and $S(p^j) = max\{v(A(p^j)), F(p^{j-1})\}; j \geq 1$. Here $v(A(p^j))$ refers to the virtual arrival time of the $j$-th packet. WFQ schedules packets in an increasing order of finish tags.

*Self Clocked Fair Queuing (SCFQ)* assings two tags, a start tag $S$ and a finish tag $F$, to each packet, $S_i(j) = max\{F_i(j-1), v(AT_i(j))\}$ and $F_i(j) = S_i(j) + l_i^j \;\; j \geq 1, F_i(0) = 0$. The virtual time at $t$, $v(t)$, is defined to be equal to the finish tag of the packet in service at time $t$. $v(t) = t$ when the server is idle. SCFQ schedules packets based on their finish tags. SCFQ is a work conserving algorithm.

## References

[1] H. Zhang and S. Keshav. Comparison of rate-based service disciplines. In *ACM SIGCOMM*, August 1991.