Packet Scheduling and Discard Mechanisms (Mecanismos de Escalonamento e de Descarte de Pacotes)

Modelação e Desempenho de Redes e Serviços

Prof. Amaro de Sousa (asou@ua.pt)

DETI-UA, 2025/2026

Packet scheduling and discard mechanisms

Consider a packet switched network. On each outgoing interface of each link of a node, there is a packet queue to temporarily store the packets to be transmitted through the link.

On the outgoing interface of each link:

Scheduling mechanism*:* decides the order by which the packets in the queue of different flows are transmitted through the link

- therefore, it imposes different average packet delays to the different flows since it defines the order of their transmission.

Packet discard mechanism*:* decides how packets of different flows are accepted into the queue when the link is occupied (with the transmission of another packet)

- therefore, it imposes different packet loss rates to the different flows since it defines which packets are discarded.

Fairness of scheduling mechanisms

When a link is congested (*i.e.*, its queue is not empty), the basic scheduling problem is:

> the assignment of a <u>scarce</u> resource to different flows with <u>equal rights</u> but with <u>different resource requirements</u>.

Ideally, the assignment of the resources should be done using the max-min fairness concept, which states that:

- the resources are assigned to the flows ordered in an increasing way by their requirements;
- none of the flows is assigned with more resources than the ones they require;
- the flows whose requirements cannot be completely satisfied are assigned with the same amount of resources.

Max-min fairness with equal rights

Consider:

- a set of flows 1, 2, ..., n with resource requirements x_1, x_2, ..., x_n ordered by their requirements ($x_1 \leq x_2 ... \leq x_n$);
- a link with capacity C.

The capacity is assigned to each flow in the following way:

- Initially all flows can be assigned with $d = C/n$
- d is lower than x_1?
  - If yes, all flows 1, 2, …, n are assigned with d
  - If not, flow 1 is assigned with x_1 and all other flows 2, 3, …, n can be assigned with $d = d + (d – x_1)/(n – 1)$
- d is lower than x_2?
  - If yes, all flows 2, 3, …, n are assigned with d
  - If not, flow 2 is assigned with x_2 and all other flows 3, 4, …, n can be assigned with $d = d + (d – x_2)/(n – 2)$
- and so on…

Example 1

Consider a link with a capacity of 128 Mbps and 4 traffic flows that want to use 8, 36, 48 and 128 Mbps. Determine the link resources that should be assigned to each flow when all flows have equal rights.

i) Flow 1 can be assigned with 128/4 = 32 Mbps.

Since flow 1 wants less than 32 Mbps, <u>flow 1 is assigned with 8 Mbps</u>. There are 32 – 8 = 24 Mbps left.

ii) Flow 2 can be assigned with 128/4 + 24/3 = 40 Mbps.

Since flow 2 wants less than 40 Mbps, <u>flow 2 is assigned with 36 Mbps</u>. There are 40 – 36 = 4 Mbps left.

ii) Flow 3 can be assigned with 128/4 + 24/3 + 4/2 = 42 Mbps.

Since flow 3 wants more than 42 Mbps, <u>flow 3 and flow 4 are assigned with 42 Mbps each</u>.

Max-min fairness with different rights

In the previous case, all flows have the same rights.

When the different flows have different rights, each flow is assigned with a weight value proportional to its right.

Then, the resources are assigned according to the weighted max-min fair principle. In this case:

- The resources are assigned to the flows ordered in an increasing way by their requirements, normalized by their weight values.
- None of the flows is assigned with more resources than the ones they require.
- The flows whose requirements are not fully satisfied are assigned a resource quantity proportional to their weight values.

Example 2

Consider a link with a capacity of 128 Mbps and 4 traffic flows that want to use 8, 36, 48 and 128 Mbps. Determine the link resources that should be assigned to each flow when the flows have the weight values 1, 1, 3 and 3, respectively.

i) Flow 1 : $1/(1+1+3+3) \times 128 = 16$ Mbps

Flow 2 : $1/(1+1+3+3) \times 128 = 16$ Mbps

Flow 3 : $3/(1+1+3+3) \times 128 = 48$ Mbps

Flow 4 : $3/(1+1+3+3) \times 128 = 48$ Mbps

<u>8 Mbps are assigned to flow 1</u> (<16 Mbps) and <u>48 Mbps to flow 3</u>.

There are $(16 – 8) + (48 – 48) = 8$ Mbps left.

ii) Flow 2 : $16 + 1/(1+3) \times 8 = 18$ Mbps

Flow 4 : $48 + 3/(1+3) \times 8 = 54$ Mbps

<u>18 Mbps are assigned to flow 2</u> (<36 Mbps) and <u>54 Mbps are assigned to flow 4</u> (<128 Mbps).

Comparison of the results of Examples 1 and 2

Link capacity: 128 Mbps

Flows:	1	2	3	4
Throughput (Mbps):	8	36	48	128
Weights:	1	1	1	1
Assigned throughput (Mbps):	**8**	**36**	42	42
Weights:	1	1	3	3
Assigned throughput (Mbps):	**8**	18	**48**	54

- When all flows have equal weights, flows 1 and 2 get their total throughput because they are the one with less throughput
- When the weight values of flows 3-4 are three times the weight values of flows 1-2, flows 3-4 get higher throughputs at the cost of flow 2 which gets less throughput than its need

Protection in scheduling mechanisms

Ideally, the scheduling mechanism should protect the good behaved flows against bad behaved flows.

- A bad behaved flow is a flow that sends more traffic than it is supposed to do (accordingly to the assigned resources).
- When good behaved flows are protected, it means that a bad behaved flow only penalizes itself concerning the quality of service provided by the network.

In the scheduling mechanisms addressed next:

- the first-in-first-out (FIFO) and the priority queueing (PQ) mechanisms do not protect good behaved flows from bad behaved flows;
- the *round-robin* based mechanisms and the mechanisms that are approximations of the GPS system protect good behaved flows from bad behaved flows.

Scheduling mechanisms

The scheduling mechanisms can be classified as work conserving and non work conserving mechanisms:

- in a work conserving mechanism, the connection is inactive (i.e., not sending any packet) only if no packet is in the queue
- in a non work conserving mechanisms, the connection might be inactive even with packets waiting in the queue

In this course unit, we only consider work conserving mechanisms. The following packet scheduling mechanisms will be addressed:

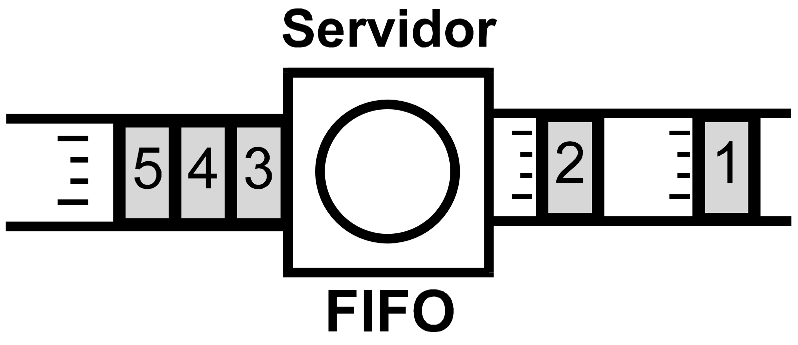
(1) by the arrival order (FIFO),

(2) based on priorities (PQ),

(3) based on round-robin (RR, WRR, DRR),

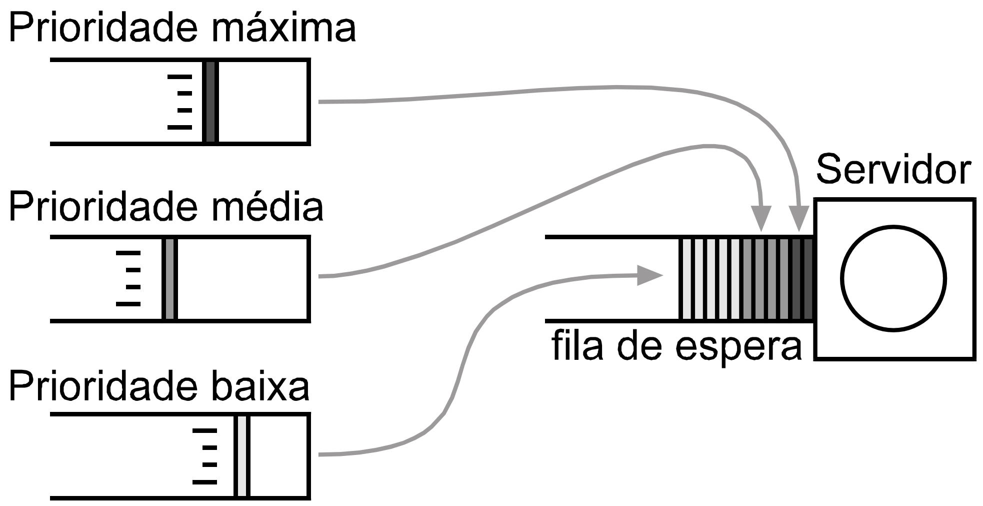
(4) by approximation to the GPS system (WFQ, SCFQ).

First-In-First-Out (FIFO)



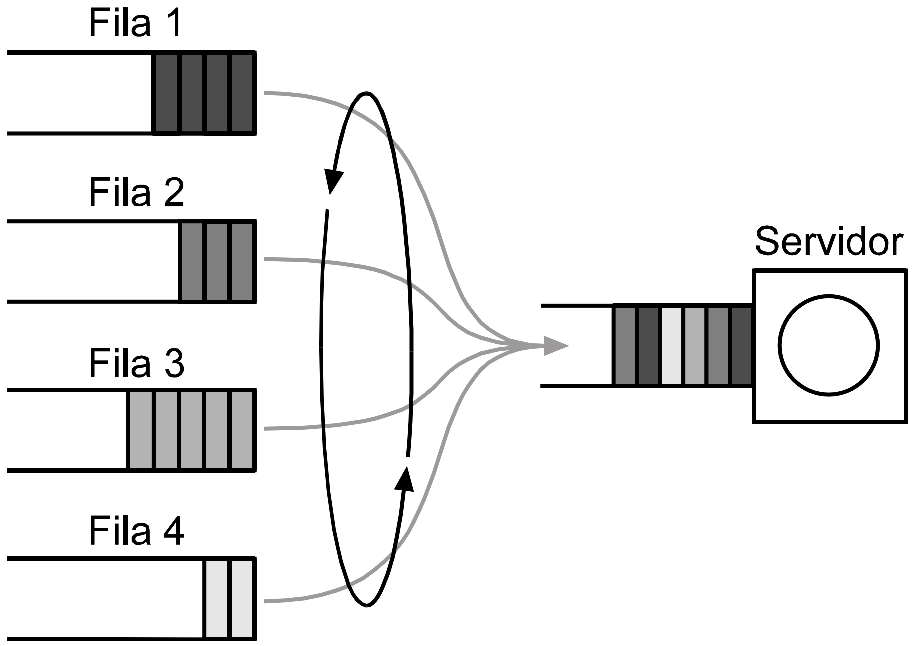
- Packets of all flows are served by their arriving order.
- It doesn’t require neither packet reordering nor packet classification.
- It does not allow quality of service differentiation between different flows (the average queuing delay is the same for all flows).
- When the queue is not empty, flows with *n* times more traffic receive *n* times more service rate
  - consequently, a bad behaved flow generating more traffic than required receives more service rate than a good behaved flow

Priority Queuing (PQ)



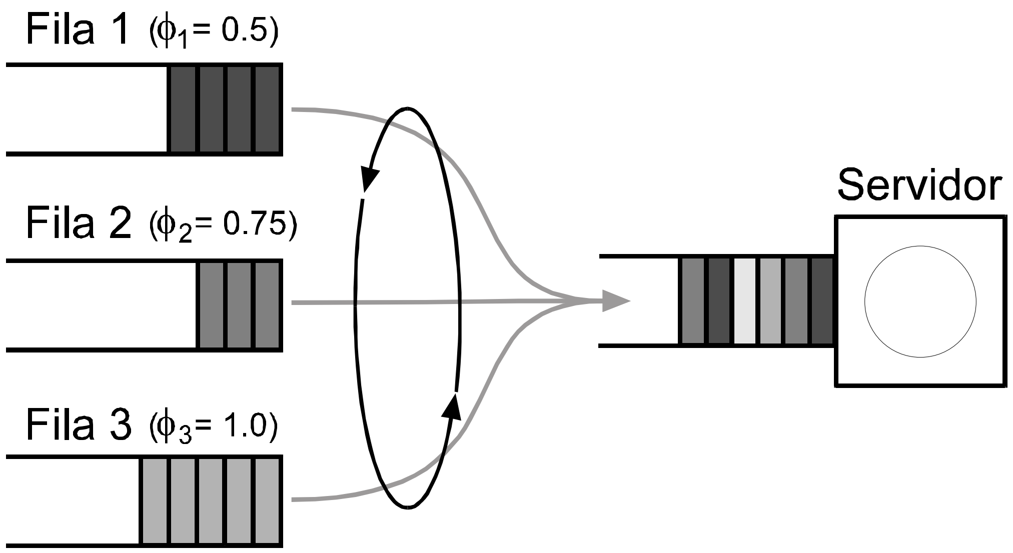
- Flows classified with a higher priority are always served before flows classified with a lower priority (packets of the same priority are served by their arriving order).
- It does not require packet reordering.
- It requires packet classification in different priorities.
- It allows quality of service differentiation between different flows (the average queuing delay is lower for higher priority flows).
- Unless prevented, it can have the starvation problem: flows of higher priority can completely prevent the other flows to be served.

Round Robin (RR)



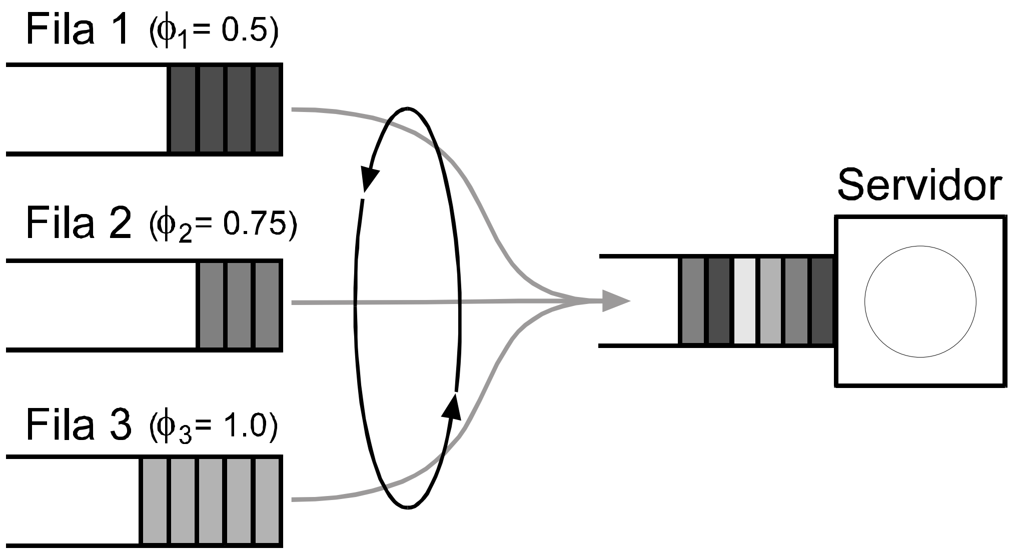
- There is a queue for each traffic flow, and the mechanism selects one packet from each nonempty queue in a round robin way.
- It does not allow quality of service differentiation.
- It gives more service rate to the flows with a higher average packet size.
- Unlike FIFO, since it serves the same number of packets among all active flows, it protects the good behaved flows from the bad behaved flows (the bad behaved flows only penalize their own average queuing delay).

Weighted Round Robin (WRR)



- A weight value ϕ_i is assigned to each queue proportional to the service rate that is to be provided to each flow.
- On each cycle, the algorithm serves a number of packets of each queue such that the sum of their sizes (in bytes) is proportional to the weight of their queue.
  - The average packet size of each flow must be known in advance.
- The connection might be too long serving the packets of each flow, which degrades the *jitter* (*i.e.*, the difference between the maximum and the minimum packet delay) suffered by each flow.

Weighted Round Robin (WRR)



On the above example, if the average packet size of each flow is (in Bytes):

$L_1 = 50,\ L_2 = 500,\ L_3 = 1500$

The normalized weights are:

$\varphi_1 = 0.5/50 = 1/100 = 60/6000$

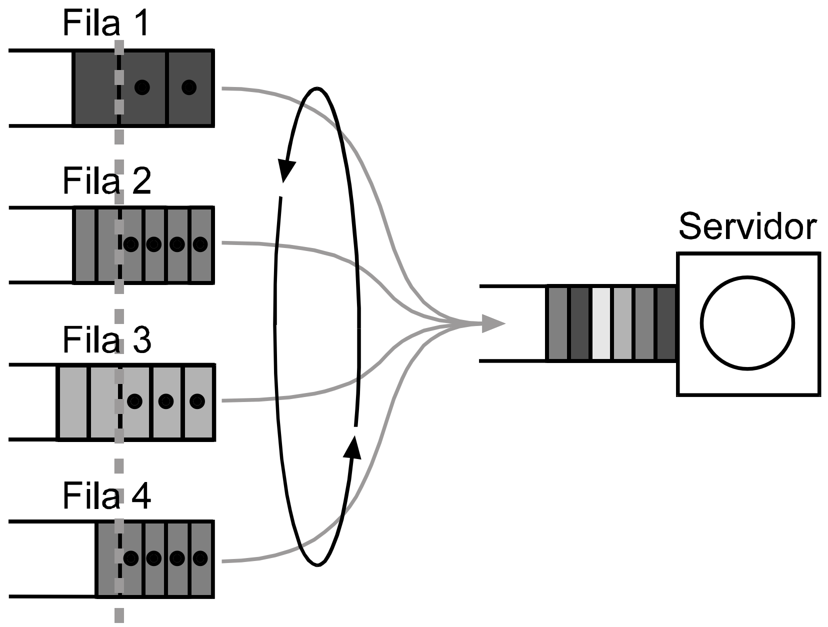
$\varphi_2 = 0.75/500 = 3/2000 = 9/6000$

$\varphi_3 = 1/1500 = 4/6000$

The maximum number of packets of each flow transmitted per cycle is:

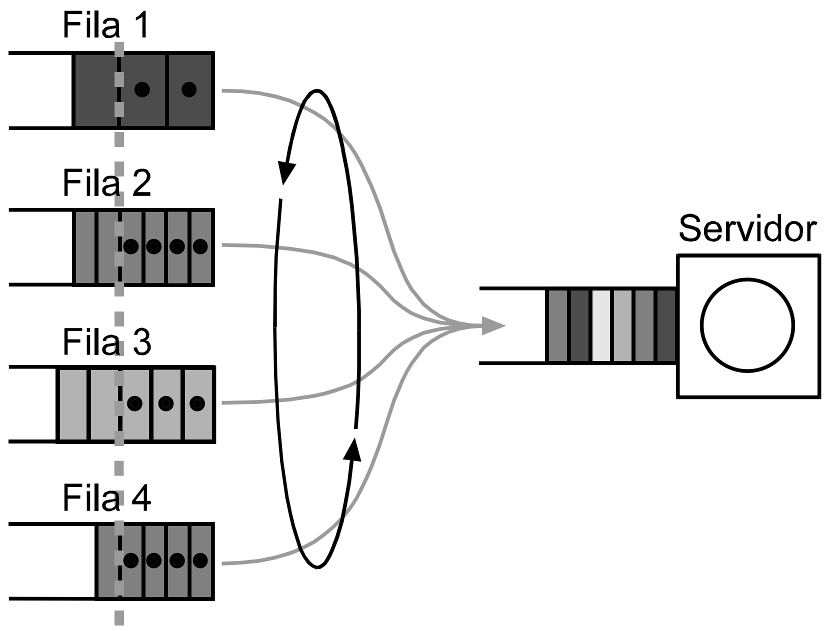
$\Phi_1 = 60,\ \Phi_2 = 9,\ \Phi_3 = 4$

Deficit Round Robin (DRR)

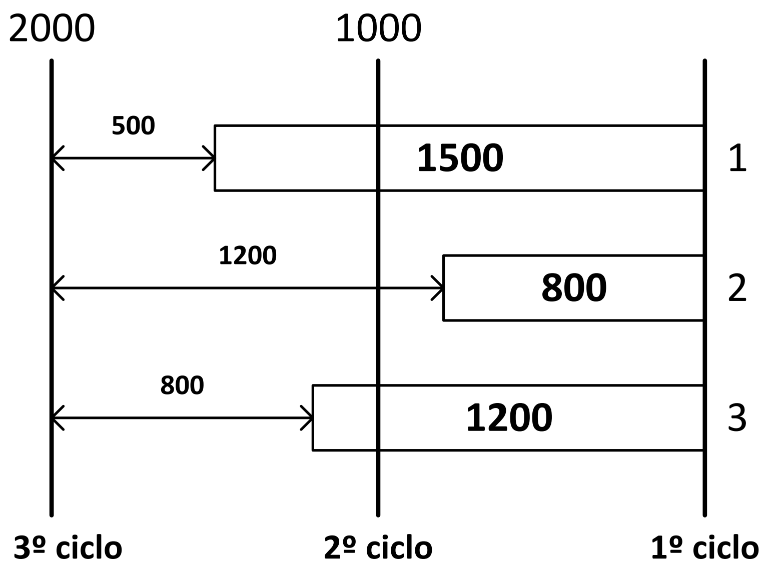


- On each cycle, the algorithm serves a quantity in Bytes up to un upper bound designated as *quantum*.
- The difference between the number of bytes served and the quantum is computed as a *credit* for the next cycle.
- When a queue is empty, its credit is set to zero.
- If we assign different quantum values to the queues, the service rate of each flow is proportional to the quantum assigned to its queue.
- As opposed to WRR, the average packet size of each flow does not need to be known in advance.

Deficit Round Robin (DRR)



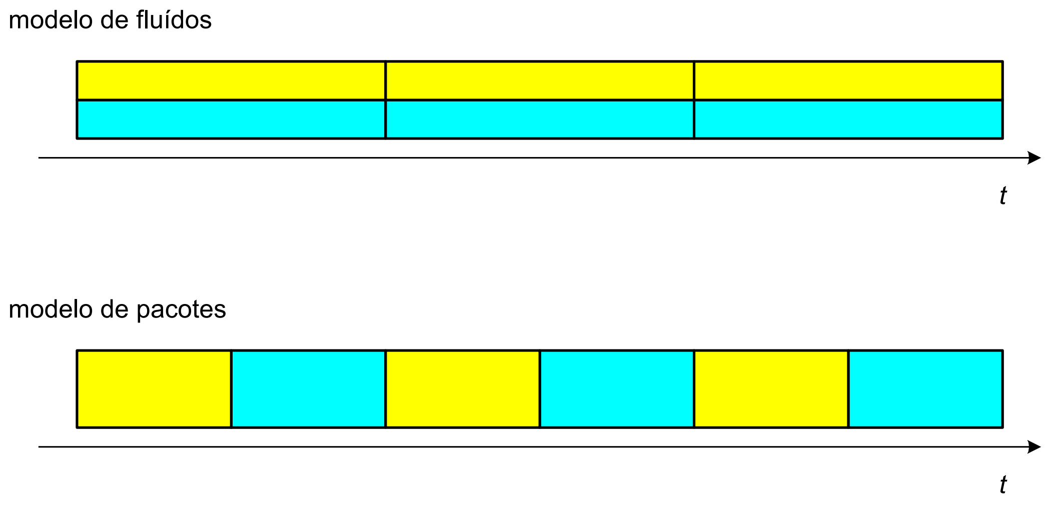
quantum = 1000 bytes (for all flows)



1st round-robin cycle:
- a) queue 1 is not served, gets a credit of 1000
- b) queue 2 is served, gets a credit of 200
- c) queue 3 is not served, gets a credit of 1000

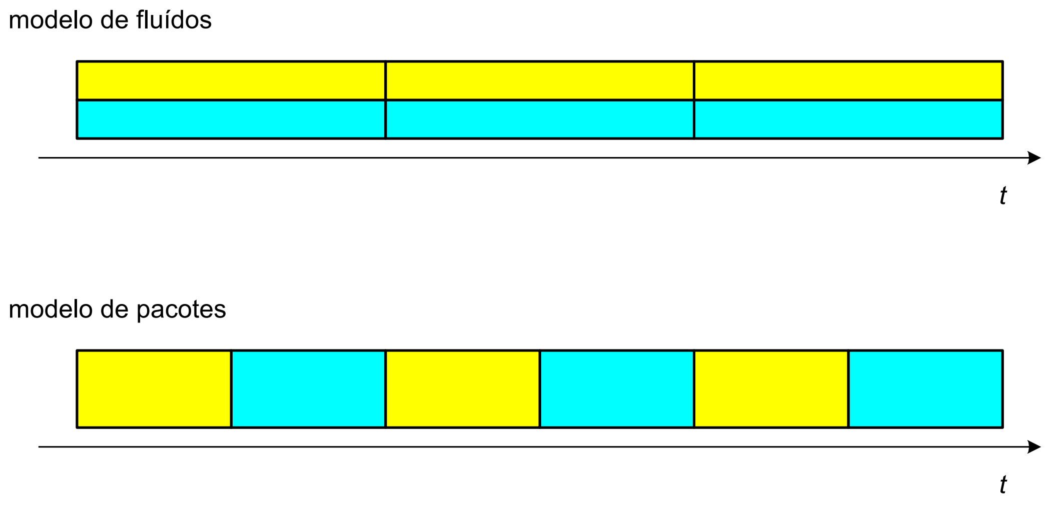
2nd round-robin cycle:
- a) queue 1 is served, gets a credit of 500
- b) queue 2 is empty, the credit becomes 0
- c) queue 3 is served, gets a credit of 800

Generalized Processor Sharing (GPS)



- This is an ideal scheduling algorithm providing perfect fairness.
- It is based on a fluid model where the traffic flows can be sent in parallel with any percentage values:
  - We can consider on a fluid model that, for example, on a given time instant 50% of the service rate of a connection is used by one flow and 50% by another flow.
- In practice, we have a packet model instead where at each time instant, the connection service rate is fully used to transmit the packet of a single flow.

Generalized Processor Sharing (GPS)



- There is a queue for each flow and a weight value ϕ_i is assigned to each queue.
- As soon as a packet arrives to a queue, if no other packet of the same flow is in the system, the packet starts being transmitted, in parallel with the packets of the other active flows, with a service rate proportional to its weight value.
- This algorithm cannot be realized in practice. Practical scheduling mechanisms try to approximate their behavior to GPS, and this algorithm is used as a reference to evaluate their performance.

Weighted Fair Queuing (WFQ)

- WFQ is an approximation to the ideal GPS system:
  - it tries to serve the packets of the different flows by the order they would end being transmitted on the GPS system.
- When a packet arrives to a queue, a ***Finish Number*** (*FN*) is assigned to it with an estimation of the time the packet would finish being transmitted in the GPS system.
- To compute the *FN* of each arriving packet, an auxiliary variable is used, named ***Round Number*** (*RN*).
  - *RN* is a real variable that increases in time with a rate inversely proportional to the weights of the active flows.
  - On an interval of time $[\tau_i, \tau_{i+1})$ such that the number of active flows remains constant, the *RN* is:

$$RN(\tau_i + t) = RN(\tau_i) + \frac{1}{\sum\limits_{j\ ativos} \phi_j} t \qquad\qquad t \in [\tau_i, \tau_{i+1})$$

Weighted Fair Queuing (WFQ)

- The *RN* is computed in all time instants such that the number of active flows changes:
  - when a packet arrives and the queue it belongs does not have any packet in the system;
  - when a packet ends being transmitted and the queue it belongs has no other packet in the queue.
- When the *k*th packet of queue *i* (where L_k is its length) arrives, it is assigned with the finish number $FN_{i,k}$ given by:

$$FN_{i,k} = \max\left(FN_{i,k-1}, RN\right) + \frac{L_k / C}{\phi_i}$$

  where C is the capacity of the link.
- When the transmission of a packet finishes, the packet in the queue with the lowest *FN* starts being transmitted.

Self Clock Fair Queuing (SCFQ)

- The main disadvantage of WFQ is the computational complexity of the calculation of the Round Number (*RN*).
- In order to get a lower complexity algorithm, in SCFQ, the RN variable is replaced by the *FN* value of the packet being in transmission, FN_s, whatever flow it belongs.
- In this way, the $FN_{i,k}$ assigned to the k^{th} packet, with length L_k, that arrives to queue *i* is given by:

$$FN_{i,k} = \max\left(FN_{i,k-1}, FN_s\right) + \frac{L_k}{\phi_i}$$

  the link capacity *C* is not used since there is no need to determine when the packet would end to be transmitted in the GPS system.
- Although the SCFQ algorithm is much less complex than WFQ, it might be not as fair as WFQ for small time intervals (i.e., its behaviour is not as close to GPS as WFQ).

Example 3

Consider a link of 64 Kbps with 2 queues with weights $\phi_1 = 3$ and $\phi_2 = 1$. Consider that the following packets arrive to the link:

- packet 1 of queue 1 with 62 Kbits (t = 0 sec.)
- packet 1 of queue 2 with 32 Kbits (t = 4 msec.)
- packet 2 of queue 1 with 18 Kbits (t = 6 msec.)

Determine and justify the time instants at which each packet finishes to be transmitted by the link when the 2 packet flows are served by:

(a) the WFQ mechanism

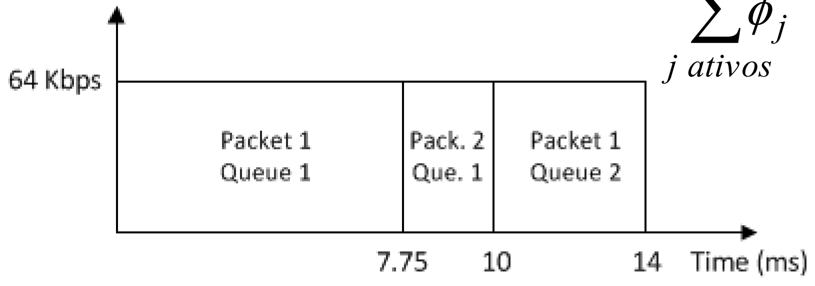
(b) the SCFQ mechanism

Example 3 – resolution of (a)

$$FN_{i,k} = \max\left(FN_{i,k-1}, RN\right) + \frac{L_k / C}{\phi_i}$$

$$RN(\tau_i + t) = RN(\tau_i) + \frac{1}{\sum_{j\ ativos} \phi_j} t$$

Link: 64 Kbps
2 queues: ϕ_1 = 3 e ϕ_2 = 1
Arrive:
- packet 1 to queue 1 with 62 Bytes (*t* = 0),
- packet 1 to queue 2 with 32 Bytes (*t* = 4 ms)
- packet 2 to queue 1 with 18 Bytes (*t* = 6 ms)

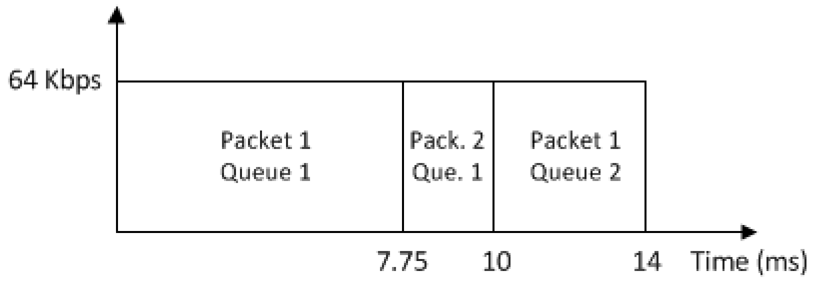


- In t = 0 ms, RN = 0 e $FN_{1,1} = 0 + (62\times8)/64000/3 = 2.58\times10^{-3}$. Packet 1 of queue 1 is transmitted in (62×8)/(64Kb/s) = 7.75 ms. So, <u>the transmission of packet 1 of queue 1 finishes at $t = 0 + 7.75 = 7.75$ ms</u>.
- In t = 4 ms : $RN = 0 + (4\times10^{-3})/3 = 1.33\times10^{-3}$
  $FN_{2,1} = 1.33\times10^{-3} + (32\times8)/64000/1 = 5.33\times10^{-3}$
- In t = 6 ms : $RN = 1.33\times10^{-3} + (6\times10^{-3} - 4\times10^{-3})/4 = 3.33\times10^{-3}$
  $FN_{1,2} = \max(2.58\times10^{-3}, 3.33\times10^{-3}) + (18\times8)/64000/3 = 4.08\times10^{-3}$
- In t = 7.75 ms, since $FN_{1,2} < FN_{2,1}$, packet 2 of queue 1 starts being transmitted. Packet 2 of queue 1 is transmitted in (18×8)/(64Kb/s) = 2.25 ms. So, <u>the transmission of packet 2 of queue 1 finishes at $t = 7.75 + 2.25 = 10$ ms</u>.
- In t = 10 ms, packet 1 of queue 2 starts being transmitted. Packet 1 of queue 2 is transmitted in (32×8)/(64Kb/s) = 4 ms. So, <u>the transmission of packet 1 of queue 2 finishes at $t = 10 + 4 = 14$ ms</u>.

Example 3 – resolution of (b)

$$FN_{i,k} = \max\left(FN_{i,k-1}, FN_s\right) + \frac{L_k}{\phi_i}$$

Link: 64 Kbps
2 queues: $\phi_1 = 3$ e $\phi_2 = 1$
Arrive:
- packet 1 to queue 1 with 62 Bytes (*t* = 0),
- packet 1 to queue 2 with 32 Bytes (*t* = 4 ms)
- packet 2 to queue 1 with 18 Bytes (*t* = 6 ms)



- In $t = 0$ ms, $FN_{1,1} = 0 + (62\times8)/3 = 165.3$. Packet 1 of queue 1 is transmitted in $(62\times8)/(64\text{Kb/s}) = 7.75$ ms. So, the transmission of packet 1 of queue 1 finishes at $t = 0 + 7.75 = 7.75$ ms.
- In $t = 4$ ms : $FN_{2,1} = 165.3 + (32\times8)/1 = 421.3$
- In $t = 6$ ms : $FN_{1,2} = \max(165.3, 165.3) + (18\times8)/3 = 213.3$
- In $t = 7.75$ ms, since $FN_{1,2} < FN_{2,1}$, packet 2 of queue 1 starts being transmitted. Packet 2 of queue 1 is transmitted in $(18\times8)/(64\text{Kb/s}) = 2.25$ ms. So, the transmission of packet 2 of queue 1 finishes at $t = 7.75 + 2.25 = 10$ ms.
- In $t = 10$ ms, packet 1 of queue 2 starts being transmitted. Packet 1 of queue 2 is transmitted in $(32\times8)/(64\text{Kb/s}) = 4$ ms. So, the transmission of packet 1 of queue 2 finishes at $t = 10 + 4 = 14$ ms.

Packet Discard Methods

- The packet discard method decides how arriving packets of different flows are either discarded or accepted into the queue when the link is occupied (with the transmission of another packet).
- The packet discard method can be used to differentiate the quality of service provided to different flows in terms of packet loss rate.
- The packet discard methods can be characterized with the following characteristics:
  - The discarding position
  - The discarding priority
  - The aggregation degree
  - Early discard versus overload discard

Packet Discard Methods

The discarding position (I)

- Tail drop – this is the default strategy and the easiest to implement: the arriving packet is the one that is discarded if it cannot be accommodated in the queue.
  - Most flows are bursty in current networks, which means that many times, the queue is full of packets of only a few flows.
  - If the arriving packet is from another flow, tail drop is not fair (to be fair, the discarded packet should be from the flow with the highest number of queued packets).
- Random drop – a packet (between the arriving one and the ones in the queue) is randomly selected to be discarded.
  - It is computationally heavy due to the random number generation.
  - Flows with more packets in the queue have more probability of suffering packet discard, which is a fairer strategy.

Packet Discard Methods

The discarding position (II)

- Head drop – the first (i.e., oldest) packet in the queue is discarded and the arriving packet is inserted at the end of the queue.
  - The probability of affecting a flow is equal when selecting a random packet or selecting the oldest packet (same fairness as random drop).
  - Head drop is not as complex as random drop since it does not need to compute random numbers.
  - Discarding the oldest packet is useful when flows are controlled by the TCP flow control algorithm: the sources of the flows suffering packet discard will lower their sending rate and, so, discarding the oldest packet makes this mechanism to react in shorter times.

Packet Discard Methods

The discarding priority

- A policy entity at an entry network node can mark the packets of some flows with a higher priority discarding value. When a link is congested, these packets are first discarded.
- When a packet is fragmented and one of the fragments is discarded in a congested link, the remaining fragments can (and must) also be discarded as soon as possible (otherwise, they will be discarded at the destination).
    - not used in IP networks since flows can be routed through different paths and both the flag ‘*more fragments*’ and the field *Fragment Offset* belong to the TCP header, whose processing is outside the scope of the routers.
- A possible strategy is to discard the packets routed by a lower number of links (they have used less resources).
    - cannot be used in IP networks because there is no header field on IP packets providing this information.

Packet Discard Methods

The aggregation degree

- The packet discard method can be applied to individual flows or to flow aggregates:
  - In the aggregated form, the flows belonging to the same aggregate are treated as a group and the discarding decision does not distinguish the individual flows.
  - Therefore, among the flows of the same aggregation, there is no packet loss rate differentiation of individual flows.
- If packets are queued by flow and the queuing memory is shared among all queues:
  - a max-min fair memory allocation is obtained if we discard the last packet of the longest queue (i.e., of the flow with the highest number of queued packets).
  - This is equivalent, in WFQ, to discard the packet with the highest *FN* (Finish Number) among all flows.

Packet Discard Methods

Early discard versus overload discard (I)

Packet discard might be applied:

- when the queues are full (overload discard)
- or before the queues become full (early discard)

In overload discard:

- when a queue is full for a significant amount of time (i.e., the link is heavily congested), packets of different TCP flows are discarded almost at the same time;
- in this case, the congestion control of the affected TCP flows reacts almost simultaneously lowering their sending data rate;
- the result is that the traffic might have an undesirable cyclic behavior between low traffic periods and heavy traffic periods.

Packet Discard Methods

Early discard versus overload discard

Random Early Discard (RED):

- When a packet arrives, it is discarded with a probability proportional to the queue occupation
- It avoids the TCP congestion control synchronism between different TCP flows.
- It does not provide quality of service differentiation in terms of packet loss rate.

Weighted Random Early Discard (WRED):

- Different discard probabilities are assigned to different packet flows (or flow aggregates).
- Decreasing the packet discard probability also decreases (i.e., improves) the packet loss rate of the flow (or flow aggregate).

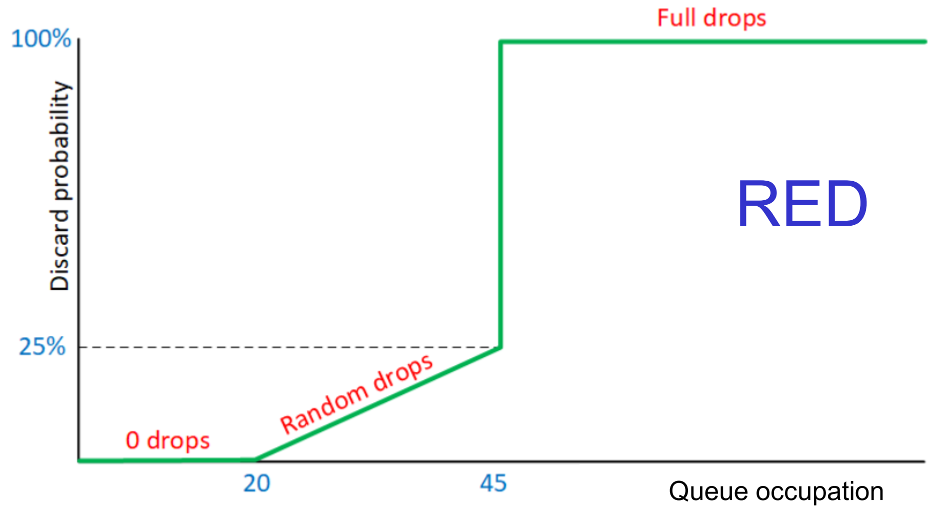
RED and WRED

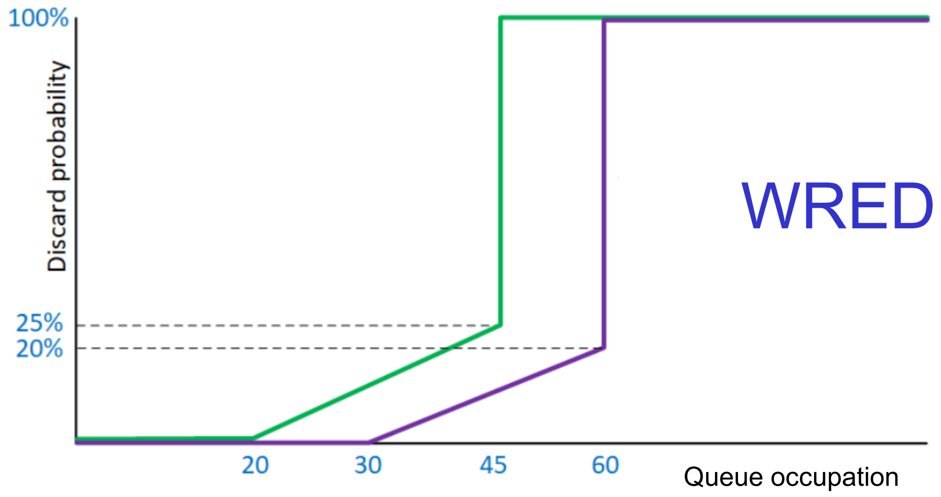
In RED:

- *Minimum Threshold (m)*: when a packet arrives and the queue occupation *f* is lower than the minimum threshold ($f < m$), the packet is always accepted.
- *Maximum Threshold (M)*: when a packet arrives and the queue occupation *f* is higher than the maximum threshold ($f > M$), the packet is always discarded.
- *Mark Probability Denominator (MPD)*: when a packet arrives and the queue occupation *f* is between the minimum *m* and maximum *M* thresholds, the packet is accepted with a drop probability $(f–m)/(M–m)\times MPD$

In WRED:

- Different *m*, *M* and *MPD* parameters are assigned to different flows (or flow aggregates)





Figures adapted from:
https://networklessons.com/cisco/ccie-routing-switching-written/wred-weighted-random-early-detection