Active Queue Management

- Active Queue Management (AQM) is a feature that can be added to a buffer, in order to manage efficently the packet dropping process
- In fact, queues have a necessarily finite storage capacity and in case of congestion they may drop packets
- In a queue without AQM, packets are *taildropped*, meaning that as soon as the queue is full and a new packet arrives, the packet is dropped
- This method has served the Internet well for years, but it has two important drawbacks:
  - Lock-out
  - Full queues

Lock out

- In some situations, especially with TCP, tail drop allows a single connection or a few flows to monopolize queue space, preventing other connections from getting room in the queue
- This "lock-out" phenomenon is often the result of TCP synchronization
- Lock out is a serious problem that creates a significant unfairness in the basic Best Effort service
- Active Queue Management is a means to cope with the lock out phenomenon

Full queues

- The tail drop discipline allows queues to maintain a full (or, almost full) status for long periods of time, since tail drop signals congestion (via a packet drop) only when the queue has become full
- Reducing the steady-state queue size is one of the most important objectives of AQM
- Even though TCP constrains a flow's window size, packets often arrive at routers in bursts
- If the queue is full or almost full, an arriving burst will cause multiple packets to be dropped
- This can result in a global synchronization of flows throttling back, followed by a sustained period of lowered link utilization, reducing overall throughput

Full queues

- Buffering absorbs data bursts and to transmit them during the ensuing bursts of silence
- This is essential to permit the transmission of bursty data
- Queue capacity must be used to absorb the bursts
- Maintaining normally-small queues can result in higher throughput as well as lower end-to-end delay than keeping queues almost full
- The limits on queue occupancy do not reflect the steady state queues we want maintained in the network
- Instead, limits on queue occupancy reflect the size of bursts we need to absorb

Active Queue Management

- The solution to the full-queues problem is for routers to start dropping packets before a queue becomes full, so that end nodes can respond to congestion before buffers overflow
- This proactive approach is called Active Queue Management
- By dropping packets before buffers overflow, active queue management allows routers to control when and how many packets to drop
- A simple Active Queue Management technique is the Random Early Detection (RED)

Random Early Detection

- Random Early Detection, or RED, is an active queue management algorithm for routers
- The RED algorithm drops arriving packets probabilistically
- The probability of drop increases as the estimated average queue size grows
- Note that RED responds to a time-averaged queue length, not an instantaneous one
- Thus, if the queue has been mostly empty in the "recent past", RED won't tend to drop packets (unless the queue overflows)
- On the other hand, if the queue has recently been relatively full, indicating persistent congestion, newly arriving packets are more likely to be dropped

Random Early Detection

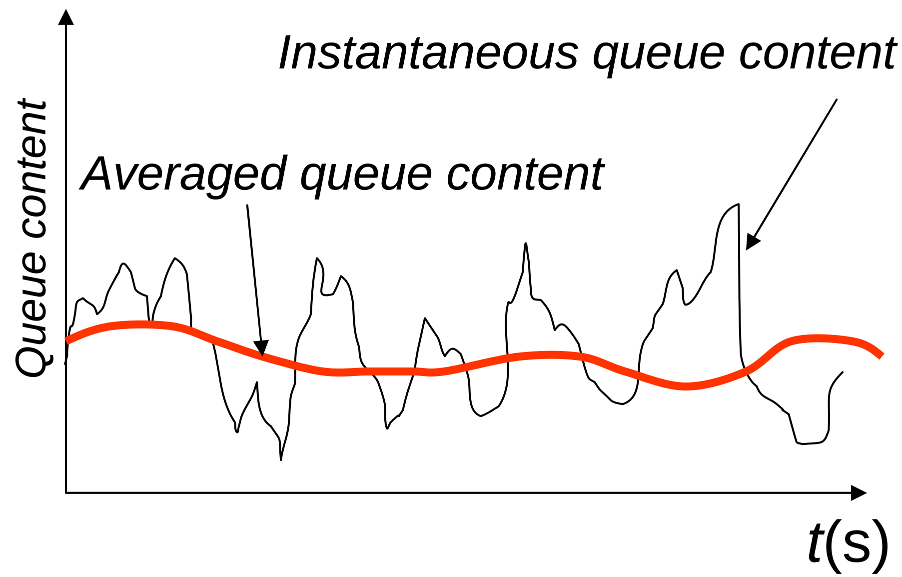
- The RED algorithm itself consists of two main parts:
  - **The estimation of the average queue size**: RED estimates the average queue size using a simple exponentially weighted moving average
  - **The decision of whether or not to drop an incoming packet**:
    - RED decides whether or not to drop an incoming packet
    - It is RED's particular algorithm for dropping that results in performance improvement for responsive flows
    - Two RED parameters, *minth* (minimum threshold) and *maxth* (maximum threshold), drive the decision process
    - *minth* specifies the average queue size *below which* no packets will be dropped
    - *maxth* specifies the average queue size *above which* all packets will be dropped
    - As the average queue size varies from *minth* to *maxth*, packets will be dropped with a probability that varies linearly from 0 to *maxp*

Random Early Detection

- The calculation of the averaged queue content is performed as follows
- At the arrival of a packet at time *t*:

$$Q_{ave} = (1-w)Q_{ave} + wQ(t)$$

- Where Q_{ave} is the estimator of the average queue occupancy, $Q(t)$ is the instantaneous queue occupancy at time t, and w is a weight in $0 < w < 1$

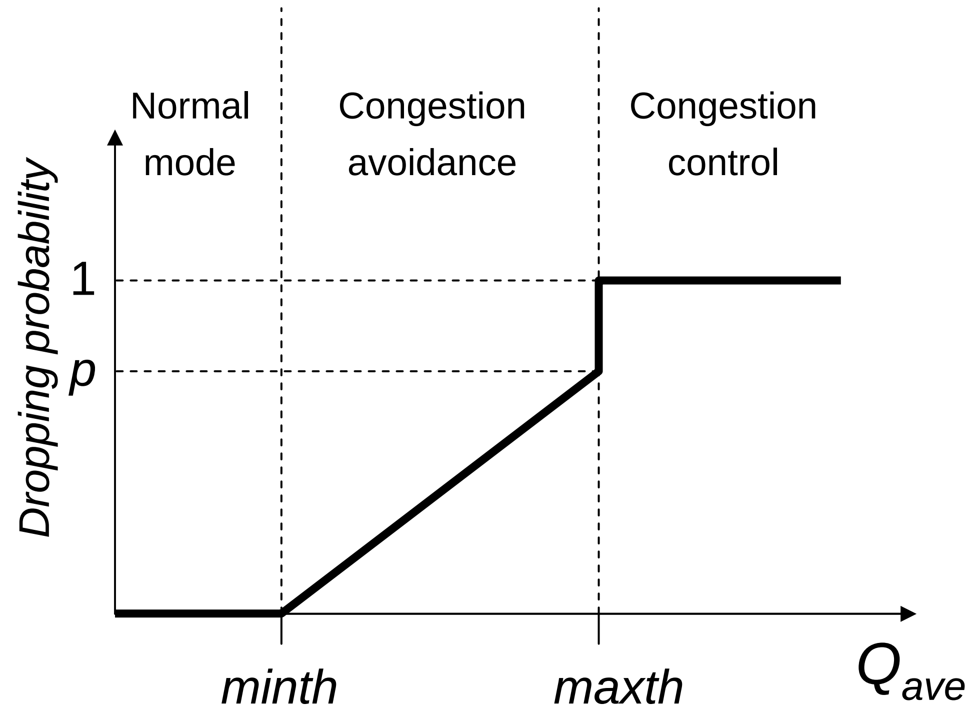


Random Early Detection

- The calculation of the averaged queue content is performed as follows
- At the arrival of a packet at time *t*:

$$Q_{ave} = (1-w)Q_{ave} + wQ(t)$$

- Where Q_{ave} is the estimator of the average queue occupancy, $Q(t)$ is the instantaneous queue occupancy at time *t*, and *w* is a weight in $0 < w < 1$
- When a packet arrives, the current value of Q_{ave} is used to determine the dropping probability of the packet, according to the plotted curve



Paolo Giacomazzi

RIO (RED for In and Out)

- When packets have different dropping priorities, the RED mechanism is insufficient as it does not differentiate packet dropping
- Thus, a more advanced mechanism is required, as in the IP Differentiated Services architecture diferent levels of packet dropping are required
- RIO (RED for In and Out) is capable of differentiating two priorities of packet dropping
- Conventionally, in RIO these two levels are referred to IN and OUT traffic
- IN traffic has a lower packet dropping probability than OUT trafifc, in the same conditions

RIO (RED for In and Out)

- RIO monitors two estimators of queue occupancy: $Q_{ave,IN}$ and $Q_{ave,OUT}$
- In particular:

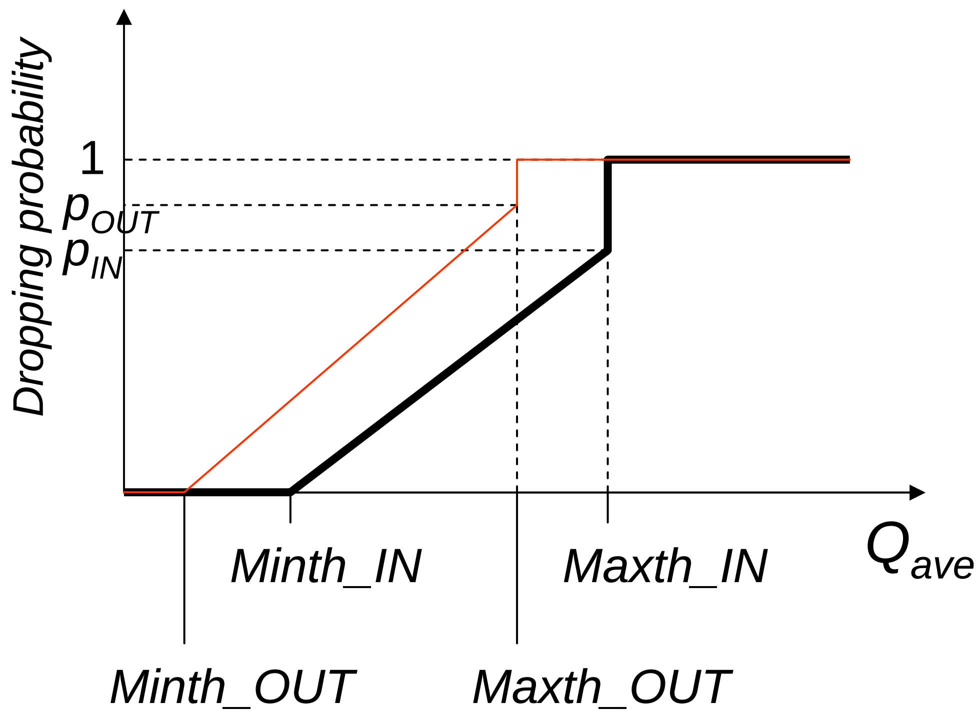
$$Q_{ave,IN} = (1 - w_{IN})Q_{ave,IN} + w_{IN}Q_{IN}(t)$$

$$Q_{ave,OUT} = (1 - w_{OUT})Q_{ave,OUT} + w_{OUT}Q_{TOT}(t)$$

- The $Q_{ave,IN}$ estimator accounts only for the queue of IN packets
- The $Q_{ave,OUT}$ and estimator accounts for both In and OUT packets
- In this way, $Q_{ave,IN}$ >= $Q_{ave,OUT}$

RIO (RED for In and Out)

- There are two sets of thresholds, one for IN and one for OUT traffic
- The thresholds are set in such a way that the dropping curve of OUT traffic is always higher than the dropping curve of IN traffic
- In this way, the dropping probability is differentiated among two classes
- This can be extended easily to an arbitrary number of dropping classes
- Usually, in the IP Differentiated Services architecture, the maximum number of different dropping behaviors is equal to three, inside a PHB group



Paolo Giacomazzi