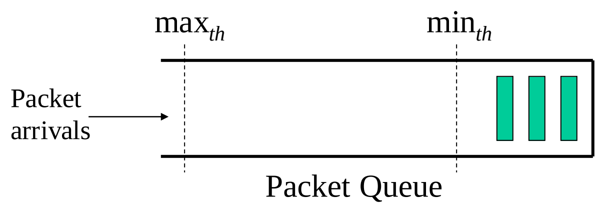
IP-Layer Congestion Control

Packet Dropping Policy

- When a packet of arrives, the packet dropping algorithm is invoked to determine if the packet is placed in queue *j*, waiting for transmission, or discarded.
- **Tail Dropping**: a packet is dropped if the queue is full when it arrives.
  - cause synchronized TCP backoff and restart
  - traffic fluctuation

Random Early Detection (RED)

- Drop *some* packets when queue start to grow.
- Start dropping packets with low probability when the average queue length *avg* reaches $\min_{th}$.
- The larger the *avg*, the more likely a packet is dropped.
- All packets are dropped when $avg \geq \max_{th}$.



Advantages

- Upon early indications of congestion, RED drops a subset of incoming packets.
- Likely, these packets belong to a subset of flows, which in turn cut their cwnd.
- This reduces the total traffic rate and prevents severe congestion from happening.

- Is it fair to dampen only a subset of flows ?

- Alternatively, TD cuts *all* flows at least by half.
  - Fair but the link will be under utilized.
  - Why can't we cut the rates of all flows by "right" amounts to produce the right total rate ?

Average Queue Length

- Upon each packet arrival,

$$avg = avg*(1-w)+w*q$$

  - w is a constant in (0,1)
  - q is the current queue length.
- q can be measured in bytes or # of packets.
- Using this *moving exponential average*, rather than the real queue length, RED accommodates temporary bursts of traffic without dropping packets.
- This is important because TCP traffic usually comes in bursts.

Accommodating Bursts

- Assume *avg*=0, and then *L* packets arrive when the transmission line is busy.

$$avg_1 = 0 \times (1-w) + w = w$$
$$avg_2 = avg_1(1-w) + 2w = w(1-w) + 2w$$
$$avg_3 = avg_2(1-w) + 3w = w(1-w)^2 + 2w(1-w) + 3w$$
$$avg_4 = avg_3(1-w) + 4w$$
$$= w(1-w)^3 + 2w(1-w)^2 + 3w(1-w) + 4w$$
$$\vdots$$

- We have

$$avg_L = w(1-w)^{L-1} + 2w(1-w)^{L-2} + \ldots + (L-1)w(1-w) + Lw$$
$$= \sum_{i=1}^{L} iw(1-w)^{L-i}$$
$$= w(1-w)^L \sum_{i=1}^{L} i\left(\frac{1}{1-w}\right)^i$$
$$= L + 1 + \frac{(1-w)^{L+1} - 1}{w}$$

- Notice that

$$\sum_{i=1}^{L} ix^i = \frac{x + (Lx - L - 1)x^{L+1}}{(1-x)^2}$$

- Given a minimum threshold $\min_{th}$ and given that we wish to allow bursts of L packets, then w should be chosen to satisfy

$$L + 1 + \frac{(1-w)^{L+1} - 1}{w} < \min_{th}$$

- For example, if $\min_{th}=5$ and $L=50$, then we must choose $w < 0.0042$.

Packet Dropping Probability

- As *avg* varies from $\min_{th}$ to $\max_{th}$, the packet dropping probability varies linearly from 0 to $\max_p$:

$$P_b = \max{}_p \times \frac{avg - \min_{th}}{\max_{th} - \min_{th}}$$

- The final packet dropping probability P_a increases slowly with the number of packets since the last dropped packet (called *count*).

$$P_a = \frac{P_b}{1 - count \times P_b}$$

The Choice of P_a

- Assuming *avg* is fixed, P_b will remain fixed too.
- Consider what happens after we have just dropped a packet (that is, count=0).
- When a new packet arrives, $P_a = P_b / (1-P_b)$.
  - Assume that we do not drop this packet.
- When the second packet arrives, $P_a = P_b / (1-2P_b)$.
- The value of P_a increases with each new packet.
- This cannot continue indefinitely.
  - When *count* $\geq (1-P_b) / P_b$, $P_a \geq 1$, forcing the packet to be dropped and resetting count to 0.

- That is, at least one packet will be dropped for every $(1-P_b) / P_b$ packets.
- Consider the probability that the *j*-th packet is dropped, where $0 < j \leq (1-P_b) / P_b$.
- We have,

$$P_a(j) = \left(1-\frac{P_b}{1-P_b}\right)\left(1-\frac{P_b}{1-2P_b}\right)\left(1-\frac{P_b}{1-2P_b}\right)\cdots\left(1-\frac{P_b}{1-(j-1)P_b}\right)\left(\frac{P_b}{1-jP_b}\right)$$

$$= \left(\frac{1-2P_b}{1-P_b}\right)\left(\frac{1-3P_b}{1-2P_b}\right)\left(\frac{1-4P_b}{1-3P_b}\right)\cdots\left(\frac{1-jP_b}{1-(j-1)P_b}\right)\left(\frac{P_b}{1-jP_b}\right)$$

$$= \frac{P_b}{1-P_b}$$

That is, $P_a(j)$ is *uniformly distributed*.

Algorithm

Initialization: avg=0, $count$=0

For each packet arrival:

If the queue is nonempty, $avg=(1-w)\times avg + wq$

Else $avg = (1-w)^m \times avg$ (decrease exponentially)

If ($min_{th} \le avg \le max_{th}$)

Count++;

$P_b = max_p (avg - min_{th}) / (max_{th} - min_{th})$

$P_a = P_b / (1 - count \times P_b)$

With probability P_a, drop this packet and set *count*=0.

Else if ($max_{th} < avg$)

drop this packet and set *count*=0

Handling Non-TCP Traffic

- Both TD and RED rely on TCP at sources to reduce traffic in the presence of congestion.
- However, some applications do not cooperate.
  - UDP applications
  - Multimedia applications
- Moreover, malicious applications deliberately refuse to cooperate.
  - Denial-of-service attacks

- A recent research area is to have the routing infrastructure punish these applications.
- A noble goal indeed !
  - The question is how to do it efficiently.
  - Per-flow monitoring places too much burden on routers (imagine a backbone router that handles thousands or more flows at a time).
- We discuss recent proposals that address this problem.

TCP Friendly Test

- A traffic flow is defined by its source address/port and destination address/port.
- A flow is **TCP-friendly** if its arrival rate does not exceed that of a conformant TCP connection in the same circumstances.
- Given a packet drop rate of *p*, packet size of *B* bytes, and round trip time *RTT*, the maximum send rate for a TCP connection is

$$B(p) \leq \frac{1}{RTT}\sqrt{\frac{3}{2p}}$$

- Packet drop rate *p* is observed by the router.
- The round trip time, however, is tricky.
  - It may vary enormously from flow to flow.
  - One solution is to use the minimum RTT seen by a router.
  - With this solution, are we being generous or harsh ?
- For a flow failing the test, a router preferentially drops its packets.
- **Limitation**: This is a per-flow mechanism and not suitable for core routers.

Responsiveness Test

- A flow is **unresponsive** (to congestion) if it does not drop its rate when the packet drop probability *p* increases.
- For example, if *p* increases by a factor of 4, a TCP conformant flow would decreases its rate by a factor of 2 according to the equation of $B(p)$.
- One (conservative) unresponsiveness test is to check, when the packet drop rate increase by a factor of 4, whether the arrival rate is decreased by at least 10%

Discussion

- Practically, we don't need to test the responsiveness of all flows, just a few high-bandwidth ones.
- However, a router still needs to spot high-bandwidth flows from a large # of flows.
  - How do you do this efficiently ?
  - Without monitoring all flows ?
  - See [HuHu03] (HuHu zero 3)

- **Proactive Testing**: Instead of applying the test passively by observing how a flow's rate changes, one can purposefully increases p for a flow and observe the changes in its packet rate.

A Stateless Solution

- All the previous tests involve flow histories.
- We will now discuss a completely stateless (no history keeping) solution.
  - When a packet arrives, randomly draw a packet from the queue.
  - If the two packets belong to the same flow, drop both; otherwise admit the new packet to the queue.
  - The basic idea is that high-bandwidth flows occupy more space in the queue and thus suffer more droppings.

Discussion

- Notice that bandwidth curbing occurs only in presence of congestion.
- Thus, aggressive applications are free to take advantages of unused bandwidth in the absence of congestion.
  - This is considered desirable.
- This solution works best when there is a small number of high-bandwidth flows.
- It does not work well when a large number of small unresponsive flows jam the network.

Analysis

- Assume that we have two flows, 1 and 2, both with Poisson arrivals with rates λ_1 and λ_2, respectively.
- The service times of the transmission line are assumed exponential, with rate μ.
- Size of the queue is assumed infinite.
- When a packet arrives, it is matched with the packet at the head of the queue.
  - This can be considered as a lazy implementation of random drawing.

- Define q_0 to be the probability that the queue is empty.
- Define q_1 and q_2 to be the probabilities that the queue head is a flow 1 and flow 2 packet, respectively.
- Define $P_{i,j}$ to be the probability that a new packet of flow i sees the head of the queue to be of type j (type 0 means empty queue).
- We claim that $P_{i,j} = q_j$.
  - This is due to a property called **PASTA**.
  - Poisson Arrivals see Time Averages

- For each flow 1 arrival, it causes $2P_{1,1}$=$2q_1$ flow 1 packets to be dropped.
  - Thus, the rate at which flow 1 packets reach the server is $\lambda_1(1-2q_1)$.
- The departure rate of flow 1 is μq_1.
  - Can you see why ?
- Since flow 1 must have identical arrival and departure rates,

$$\mu q_1 = \lambda_1(1-2q_1) \quad \Rightarrow \quad q_1 = \frac{\lambda_1}{\mu + 2\lambda_1}$$

- Following the same logic, we have

$$\mu q_2 = \lambda_2(1-2q_2) \quad \Rightarrow \quad q_2 = \frac{\lambda_2}{\mu + 2\lambda_2}$$

Discussion

- A surprising observation is that the drop rate of flow *i* (i.e., $2P_{i,i}$=$2q_i$) depends only on μ and λ_i; it has nothing to do with other flows at all.
- Thus our results generalize to any number of flows.
- Also, the performance of any given flow do not change due to congestion or the existence of other flows.