Active Queue Management

Mukul Goyal

Random Early Drop (RED) [floyd93]: Goals

- Main Goal: Provide congestion avoidance by controlling the average queue size.
  - Keep the average queue size low while allowing occasional bursts of packets in the queue.
- Additional goals:
  - Avoid global synchronization
  - Avoid bias against bursty traffic
  - Should be able to maintain an upper bound on the average queue size even in the absence of cooperation from transport-layer protocols.

Estimating Congestion: Average Queue Size Versus Instantaneous Queue Size

- TCP traffic is bursty => instantaneous queue size varies significantly => high instantaneous queue size may not indicate congestion in the network.
- Average queue size is calculated as exponentially weighted moving average of instantaneous queue size.
  - Avg = (1-a)*avg + a* inst
- Average Queue Length = lowPassFilter(instant. qlen)
- RED Argument: Average queue size is a better indicator of congestion than instantaneous queue size. *Ignore high frequency noise (i.e. rapidly varying instant. Queue length)*

RED Contd: Global Synchronization and Bias Against Bursty Sources

- With DropTail gateways each congestion period introduces global synchronization in the network.
  - When the queue overflows, packets are often dropped from several connections, and these connections decrease their windows at the same time. This results in a loss of throughput at the gateway.
- DropTail gateways are likely to be biased against bursty sources.
  - A burst of packets from a source likely to cause buffer overflow leading to the drop of several packets which will significantly slow down the bursty source.

RED Contd: Congestion Avoidance Goals

- Detect the incipient congestion.
- Maintain the network in a region of low delay and high throughput.
- The average queue size should be kept low, while fluctuations in the actual queue size should be allowed to accommodate bursty traffic and transient congestion.

The RED Algorithm

- The RED gateway calculates the average queue size, using a low-pass filter with an exponential weighted moving average.
- The average queue size is compared to two thresholds, a *minimum* threshold and a *maximum* threshold.
- When the average queue size is less than the minimum threshold, no packets are marked.
- When the average queue size is greater than the maximum threshold, every arriving packet is marked.
- When the average queue size is between the minimum and the maximum threshold, each arriving packet is marked with probability *pa*, where *pa* is a function of the average queue size *avg*.
- Each time that a packet is marked, the probability that a packet is marked from a particular connection is roughly proportional to that connection's share of the bandwidth at the gateway.

The RED Algorithm: Basic Description

for each packet arrival:
  calculate the average queue size *avg*
  if *minth* ≤ *avg* < *maxth*
    calculate probability *pa*
    with probability *pa*:
      mark the arriving packet
  else if *maxth* ≤ *avg*
    mark the arriving packet

The RED Operation

- The RED gateway has two separate algorithms.
  - The algorithm for computing the average queue size determines the degree of burstiness that will be allowed in the gateway queue.
  - The algorithm for calculating the packet-marking probability determines how frequently the gateway marks packets, given the current level of congestion.

Detailed RED Operation: The variables

Saved Variables:
- *avg*: average queue size
- *q_time*: start of the queue idle time
- *count*: packets since last marked packet

Fixed parameters:
- w_q: queue weight
- *minth*: minimum threshold for queue
- *maxth*: maximum threshold for queue
- max_p: maximum value for p_b

Other:
- p_a: current packet-marking probability
- *q*: current queue size
- *time*: current time
- *f* (*t*): a linear function of the time *t*

Detailed RED Operation

Initialization:

avg = 0

count = -1

For each packet arrival:

calculate the new average queue size *avg*:

if the queue is nonempty

$avg = (1-w_q)avg+w_q q$

else

$m = f(time-q\_time)$

$avg = (1-w_q)^m avg$

Detailed RED Operation

if *minth* ≤ *avg* < *maxth*

increment *count*

calculate probability *pa*:

$p_b = max_p(avg-minth)/(maxth-minth)$

$p_a = p_b/(1-count \times p_b)$

with probability p_a:

mark the arriving packet

count = 0

else if *maxth* ≤ *avg*

mark the arriving packet

count = 0

else *count* = -1

Flow RED (FRED) [lin97]

RED imposes the same loss rate on all the flows regardless of their bandwidth.

- Congestion Sensitive (TCP) flows will backoff but insensitive (UDP) flows will continue to send data at the same rate as before.
- RED is unfair to congestion sensitive flows.

Flow RED: Use per-active-flow accounting to impose on each flow a loss rate that depends on the flow's buffer use.

FRED Operation

FRED maintains the following variables:

- min_q and max_q: The goals for minimum and maximum number of packets each flow should be allowed to buffer.
- Avgcq: an estimate for average per-flow buffer count. flows with fewer than avgcq packets queued are favored over flows with more.
- $qlen_i$: a count of buffered packets for each flow i that currently has any packets buffered.
- $strike_i$: the number of times the flow i has failed to respond to congestion notification; FRED penalizes flows with high strike values.

FRED Operation: For each arriving packet P from flow i

identify and manage non-adaptive flows:

```
if (qlen_i >= max_q ||
   (avg >= maxth && qlen_i > 2*avgcq) ||
   (qlen_i >= avgcq && strike_i > 1)) {
    strike_i++;
    drop the packet;
}
```

FRED: For each arriving packet P from flow i (contd.)

operate in random drop mode:

```
if (minth <= avg < maxth ) {
    Randomly drop the packet only if (qlen_i >=
      MAX(minq , avgcq))
}
```

Balanced RED [farooq99]

- Another approach to provide fair distribution of bandwidth between congestion sensitive and insensitive flows.
- Maintain 2 variables for each flow having a packet in the buffer:
  - $qlen_i$: the number of packets of flow i in the buffer
  - gap_i: the number of packets accepted from flow i since last dropping a packet from the flow.

BRED

- Parameters:
  - l_1: minimum number of packets that a flow can have in the buffer before its packets start getting dropped with probability p1.
  - l_2: the number of packets that a flow can have in the buffer before the packets get dropped aggressively with probability p2 which is greater than p1.
  - w_m: maximum number of packets that the flow is allowed to have in the buffer.

BRED

- For each arriving packet from flow i:
  - If $qlen_i > w_m$ or the buffer is full, drop the packet.
  - If $w_m > qlen_i > l_2$ && $gap_i > l_2$, drop the packet with probability p2.
  - If $l_2 > qlen_i > l_1$ && $gap_i > l_1$, drop the packet with probability p1.
  - If $qlen_i <= l_1$, accept packet.

A Self-Configuring RED Gateway [Wu99]

- Consider a bottleneck link with capacity 10Mbps.
  - 100 TCP connections sharing the link => Per-connection bw = 100kbps => a congestion signal to one connection leads to a new load of 9.95Mbps.
  - 2 TCP connections sharing the link => Per-connection bw = 5Mbps => a congestion signal to one connection leads to a new load of 7.5Mbps.
- The max_p parameter in RED should be adjusted based on the number of connections.

A Self-configuring RED Gateway

- Too aggressive packet drops will lead to empty queues and under-utilization.
- Too lenient packet drops will lead to buffer overflows => RED reduces to droptail.
- Per-flow accounting is expensive.
  - Several schemes *estimate* the number of active flows with mixed results [sred99].
- How to set max_p so that it is neither too aggressive not too lenient?

A Self Configuring RED Gateway

- If the average queue size hovers around minth, assume maxp to be too aggressive.
- If the average queue size hovers around maxth, assume maxp to be too lenient.

A Self Configuring RED Gateway

Every time average queue *avg* is updated:

```
if ( minth < avg < maxth )
    status = Between;
if ( avg < minth && status != Below)
    status = Below;
    max_p = max_p / a;
if ( avg > maxth && status != Above)
    status = Above;
    max_p = max_p * b;
```

Adaptive RED [floyd2001]

- Almost same as the self configuring RED except that *additive increase multiplicative decrease* is used to adjust max_p rather than *multiplicative increase multiplicative decrease.*
- ARED Variables:
  - Interval: 0.5 seconds
  - Target: target value for average queue [minth + 0.4*(maxth-minth), minth + 0.6*(maxth-minth)]
  - a: min(0.01, max_p/4)
  - b: 0.9

Adaptive RED Operation

- Every interval seconds:
  - If (avg > target and max_p ≥ 0.5)
    - increase max_p: max_p += a;
  - Elseif (avg < target and max_p >= 0.01)
    - decrease max_p: max_p *= b;

Control Theoretical Approaches to Buffer Management: The PI Controller [hollot01]

- Apply control theory to develop a model for TCP and AQM dynamics [misra00]
- Simplify the TCP/AQM model to a linear system and design a Proportional Integrator controller that regulates the queue length to a target value q_{ref}.
- The PI controller uses the instantaneous samples of the queue length taken at a constant sampling frequency as its input.

The PI Controller Contd.

- The packet drop probability is:
  $p(kT) = a(q(kT)-q_{ref}) - b(q((k-1)T)-q_{ref}) + p((k-1)T)$
- The drop probability increases (decreases)
  - when the queue length is higher (lower) than the target value.
  - If the queue has grown (reduced) since the last sample.
- The sampling frequency and the other coefficients depend on link capacity, highest RTT and expected number of active flows using the link.
- The controller is designed so as to be robust even when the actual (highest RTT, active flows) are different from expected values.

Control Theoretical Approaches to Buffer Management: Random Exponential Marking (REM) [athura01]

- REM periodically updates a congestion "price" that reflects any mismatch between
  - packet arrival and departure rates at the links
  - Actual queue length and target value
- Congestion price p is given by:
  $p(t) = \max(0, p(t-1) + \gamma(\alpha(q(t)-q_{ref}) + x(t) - c))$
- Where c is link capacity, q(t) is instantaneous queue length at time t and x(t) is packet arrival rate at time t.

REM Contd.

- The packet drop probability is defined as:
  $prob(t) = 1 - \phi^{-p(t)}$ where $\phi > 1$ is a constant
- In overload situations, the congestion price increases due to rate and queue mismatch leading to higher packet drop probability.
- When congestion abates, the price goes down because mismatches are negative leading to low packet drop probability.

References

[floyd93] S Floyd, V Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Transactions on Networking, 1(4), Aug 1993.

[lin97] D Lin, R Morris, "Dynamics of Random Early Detection," SIGCOMM 1997.

[farooq99] F Anjum, L Tassiulas,"Fair Bandwidth Sharing between Adaptive and Non-adaptive Flows in the Internet," INFOCOM 1999.

[wu99] W Feng, D Kandlur, D Saha, K. Shin,"A Self-Configuring RED Gateway," INFOCOM 1999.

References

[sred99] T. Ott, T Lakshman, L Wong,"SRED: Stabilized RED," INFOCOM 1999.

[floyd2001] S Floyd, R. Gummadi, S Shenker, "Adaptive RED: An Algorithm to increase the robustness of RED's Active Queue Management," http://www.icir.org/floyd/papers/adaptiveRed.pdf 2001.

[hollot01] C. Hollot, V. Mishra, W. Gong, D. Towsley, "On Designing Improved Controllers for AQM Routers Supporting TCP Flows," INFOCOM 2001.

[mishra00] V. Mishra, W. Gong, D. Towsley, "Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows With an Application To RED," SIGCOMM 2000

[athura01] S Athuraliya, V. Li, S. Low, Q Yin, "REM: Active Queue Management," IEEE Network, Vol 15, No 3, May 2001