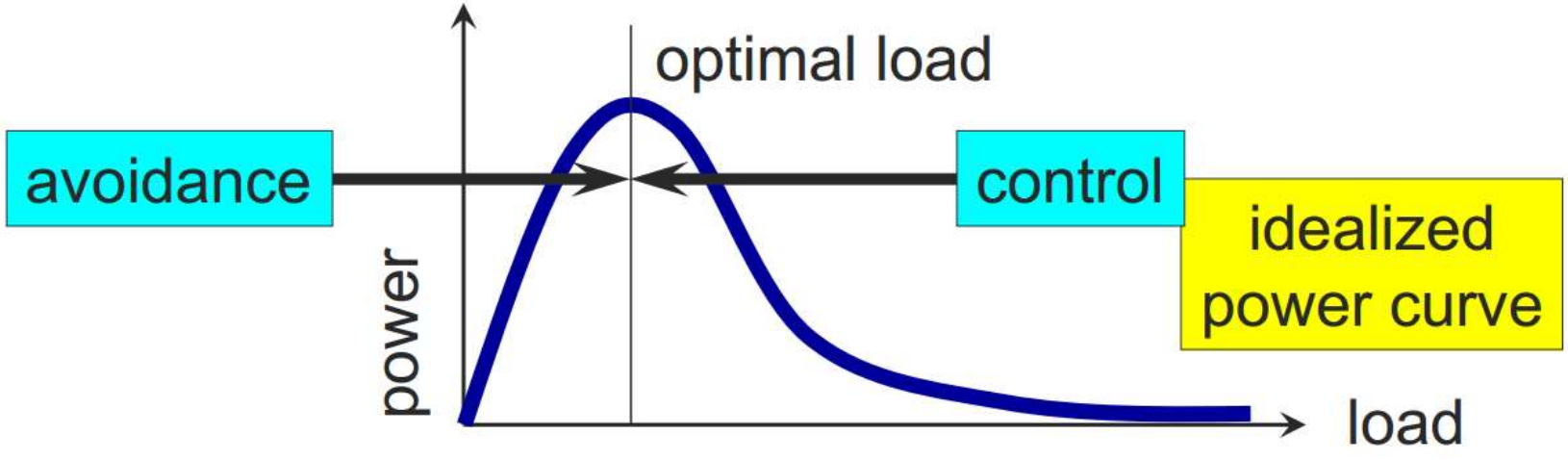
# Congestion Avoidance Techniques

# Congestion Avoidance

- Congestion control: when congestion occurs, do something to recover

- Congestion avoidance: avoid congestion before it occurs



# Congestion Avoidance

- Router-based congestion avoidance
  - RED (Random Early Detection)

- Source-based congestion avoidance
  - TCP Vegas

# Random Early Detection

- If the router is “almost” full, it starts to drop packets randomly
- TCP source reduces sending speed due to packet drop

# Random Early Detection

- Compute average queue length (AvgLen)
  - AvgLen = (1-Weight)*AvgLen + Weight*SampleLen
  - 0 < weight < 1
    - Typically, “weight” is a small value (e.g. 0.002)
  - SampleLen: the queue length when a packet arrives

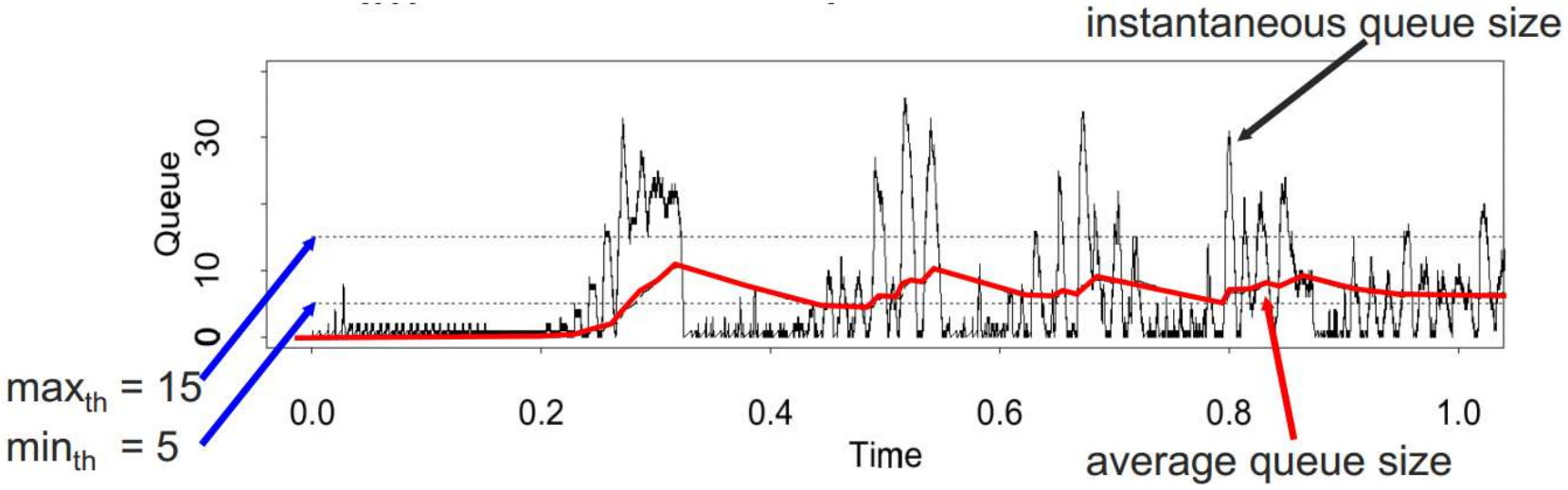
- if AvgLen ≤ MinThreshold
  - insert packet into the queue
- if MinThreshold < AvgLen < MaxThreshold
  - probabilistically drop packet (based on drop probability p)
- if MaxThreshold ≤ AvgLen
  - drop packet

# Random Early Detection

- computing average queue length: low-pass filter

$$avg \leftarrow (1 - w_q)avg + w_q\, q$$

- example: $w_q = 0.002$



# TCP Vegas

- A variation of TCP
- uses congestion avoidance instead of congestion control
- detection congestion before packet is dropped
  - if throughput does not increase much as congestion window is increased, it indicates that the network is near congestion

# TCP Vegas

- If there is no congestion, the throughput should be as expected
  - ExpectedRate: congestion window / baseRTT
  - baseRTT: minimum measured RTT
- Measure actual throughput
  - Congestion Window / currentRTT

# TCP Vegas

- Diff = ExpectedRate – ActualRate
- if Diff < a
  - Increase CWND linearly
- if Diff > b
  - Decrease CWND linearly
- else
  - Leave CWND unchanged

- decide whether CWND should be increased or decreased, based on comparison between expected rate and actual rate

# End of Class