

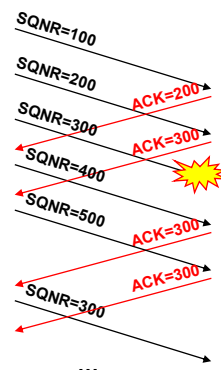
TCP's Congestion Avoidance

RED, WRED, and ECN

TCP Revisited: Duplicate ACKs



- TCP receivers send duplicate ACK if segments are missing
- ACKs are cumulative
 - ◆ Each ACK acknowledges all data until specified ACK-number



TCP Performance

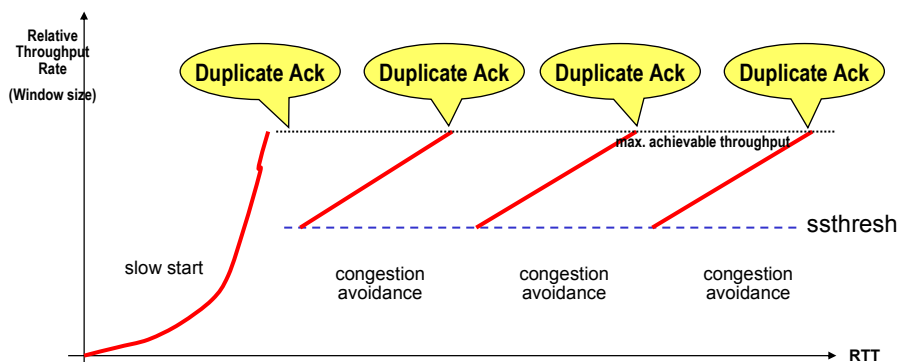


- TCP tries to minimize the data transmission time
- But good self-regulating mechanism to avoid congestion
- TCP is "**hungry but fair**"
 - ◆ But only fair to other TCP applications !
 - ◆ Problem together with voice and other realtime traffic

TCP is Hungry



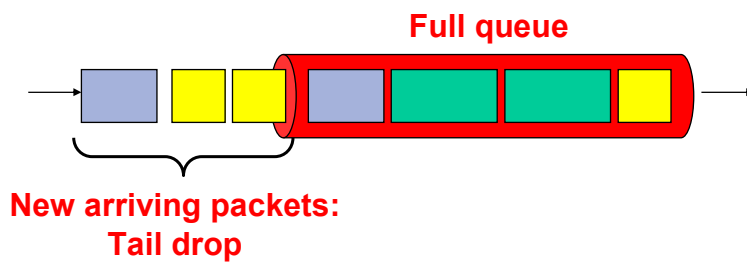
- Tries to fill the "pipe" using
 - ◆ Slow Start and
 - ◆ Congestion Avoidance



Tail-drop Queuing



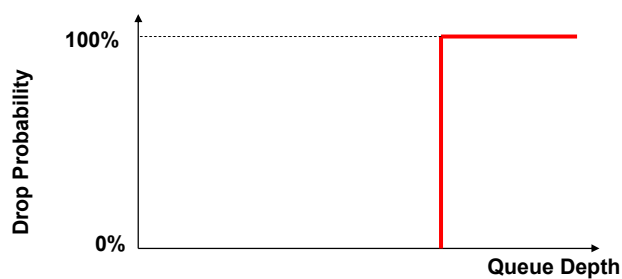
- **Standard dropping behavior in FIFO queues**
 - ♦ If queue is full all subsequent packets are dropped



Tail-drop Queuing (cont.)



- **Another representation:
Drop probability versus queue depth**



Tail-drop Problems

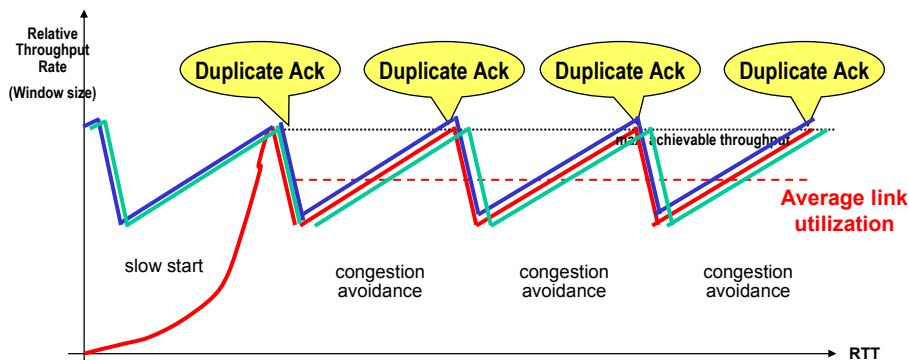


- No flow differentiation
- TCP starvation upon multiple packet drop
 - TCP receivers may keep quiet (not even send Duplicate ACKs) and sender falls back to slow start
– worst case!
 - TCP fast retransmit and/or selective acknowledgement may help
- **TCP synchronization**

TCP Synchronization



- Tail-drop drops many packets of different sessions at the same time
- All these sessions experience duplicate ACKs and perform synchronized congestion avoidance



Random Early Detection (RED)



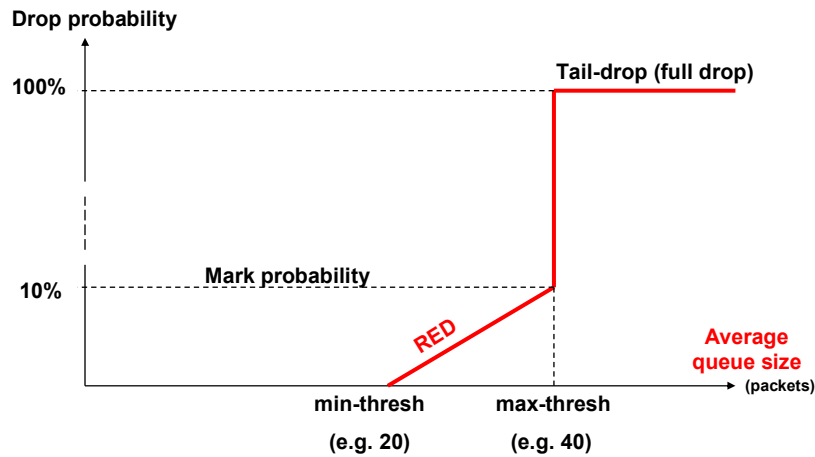
- Utilizes TCP specific behavior
 - ♦ TCP dynamically adjusts traffic throughput to accommodate to minimal available bandwidth (bottleneck) via reduced window size
- "Missing" (dropped) TCP segments cause window size reduction!
 - ♦ Idea: Start dropping TCP packets before queuing "tail-drops" occur
 - ♦ Make sure that "important" traffic is not dropped
- RED randomly drops packets before queue is full
 - ♦ Drop probability increases linearly with queue depth

RED



- Important RED parameters
 - ♦ Minimum threshold
 - ♦ Maximum threshold
 - ♦ Average queue size (running average)
- RED works in three different modes
 - ♦ No drop
 - If average queue size is between 0 and minimum threshold
 - ♦ Random drop
 - If average queue size is between minimum and maximum threshold
 - ♦ Full drop
 - If average queue size is equal or above maximum threshold = "tail-drop"

RED Parameters



Weighted RED (WRED)



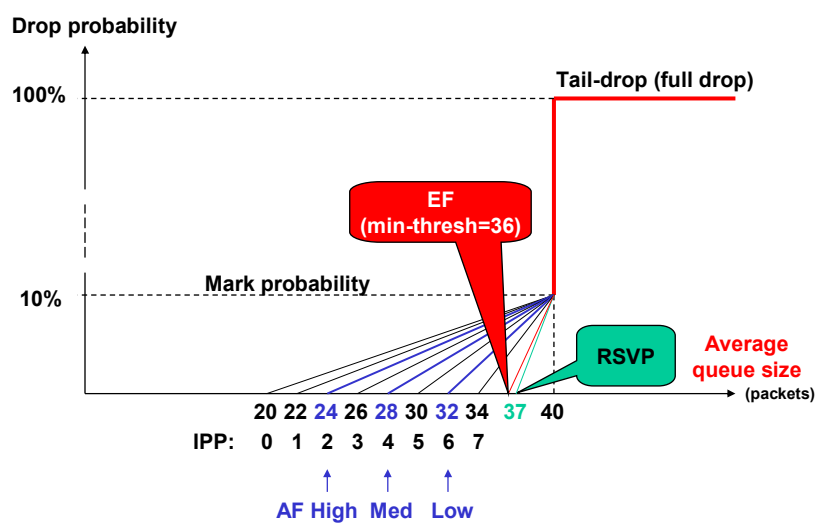
- Drops less important packets more aggressively than more important packets
- Importance based on:
 - ♦ IP precedence 0-7
 - ♦ DSCP value 0-63
- Classified traffic can be dropped based on the following parameters
 - ♦ **Minimum threshold**
 - ♦ **Maximum threshold**
 - ♦ **Mark probability denominator**
(Drop probability at maximum threshold)

RED Configuration



- Specific parameter set configurable for each DSCP or IPP value
 - ♦ Already 8 default profiles for prec-based WRED and 16 default profiles for DSCP-based WRED
- RED can be applied either
 - ♦ Directly on interface
 - ♦ Or within policy-map (part of CBWFQ configuration)
- It is recommended that RED is not enabled for EF traffic

Default Profiles



WRED - Configuration



- Enable WRED on all links where congestion might occur
 - ◆ Typically low-rate links
 - ◆ Don't apply RED for traffic that is sensitive for loss (VoIP)
- Two weighting possibilities:
 - ◆ prec-based: based on IP precedence
 - **Default setting**
 - ◆ dscp-based: based on DSCP value

```
(config-pmap-c)# random-detect [prec-based | dscp-based]
```

WRED – Configuration (cont.)



- precedence = IP precedence value
- dscp = DSCP value

- min_T = Minimum threshold
- max_T = Maximum threshold
- mpd = **Mark-prob-denominator**
 - ◆ Drop probability = 1/mpd if average queue size is at max-threshold
 - ◆ Default = 10
 - One out of 10 packets will be dropped

```
(config-pmap-c)#
```

```
random-detect precedence <precedence> <min_T> <max_T> <mpd>
```

```
random-detect dscp <dscp> <min_T> <max_T> <mpd>
```

Burst sensitivity



- WRED algorithm always *estimates* the average queue size to determine the RED mode (no/random/full drop)
 - ♦ Average Queue size running average formula:
$$Q_{avg}(t+1) = Q_{avg}(t) * (1 - 2^{-n}) + Q_t * 2^{-n}$$
- Low $n \Rightarrow$ RED becomes burst sensitive (default: $n=9$)
 - ♦ Increase n on high-speed interfaces to allow more bursts without random drops

```
(config-pmap-c)# random-detect exponential-weighting-constant <n>
```

WRED – Example



- The following example enables WRED directly on the interface and specifies parameters for the different IP precedences
- If the same profiles should be reused on other interfaces use the `random-detect group` command in global configuration

```
interface serial0/1
ip address 10.1.1.1 255.255.255.0
random-detect
random-detect precedence 0 32 256 100
random-detect precedence 1 64 256 100
random-detect precedence 2 96 256 100
random-detect precedence 3 110 256 100
random-detect precedence 4 150 256 100
random-detect precedence 5 200 256 100
random-detect precedence 6 290 256 100
random-detect precedence 7 210 256 100
!
```

WRED – Example (cont.)



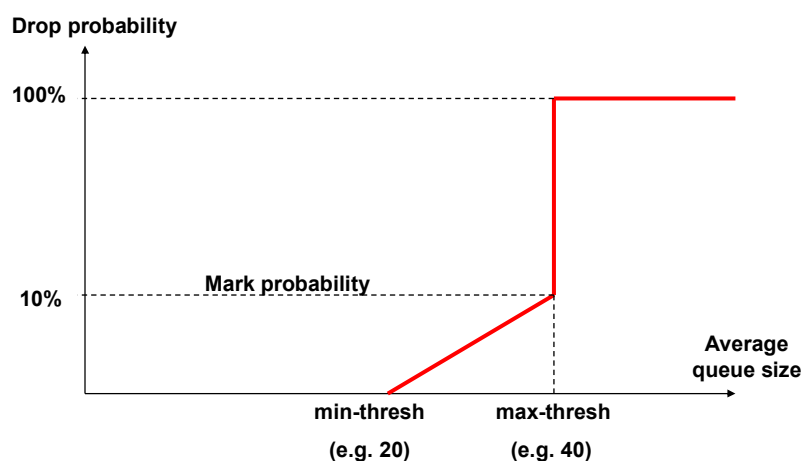
- **Note**
 - ♦ Drop probability increases linearly from zero to mark probability
 - ♦ It is recommended that RED is not enabled for EF traffic
- The default exponential weight constant is 9
- IOS marks routing protocol traffic with CS6
- ECN is set via ToS = 0x02

```
random-detect dscp <dscp-val> <min-thresh> <max-thresh>
                [<mark-probability-denominator>]
```

```
! Enable ECN
random-detect ecn
```

```
! Enable WRED or DWRED
random-detect dscp-based | prec-based
```

WRED



Flow-Based WRED



- **WRED does not differentiate between TCP segments from different flows**
 - ♦ Aggressive flows can utilize the queues exclusively and cause other flows to starve (except in true fair queuing)
- **Flow-Based WRED also keeps track of flow behavior**
 - ♦ Aggressive flows are dropped ahead of other flows

Different Flow Types



- **Robust flows**
 - ♦ React upon packet drop by reducing window-size
 - ♦ Works effective in conjunction with WRED
- **"Fragile flows"**
 - ♦ Have only a few packets in the routers buffer
 - ♦ But are dropped with same rate as aggressive flows
- **Nonadaptive flows**
 - ♦ Do not react on packet drop
 - ♦ Not effective in conjunction with WRED
 - ♦ High and constant buffer utilization

FB WRED – Configuration



Enable Flow-Based WRED

```
R1(config-if)# random-detect
R1(config-if)# random-detect flow
```

Configure maximum number of tracked flows (default = 256)

```
random-detect flow count <number>
```

Scale the queue-limit for each flow to allow some burstiness (default = 4) - Affects all queues

```
random-detect flow average-depth-factor <scaling-factor>
```

Interface configuration only!

RED Problems

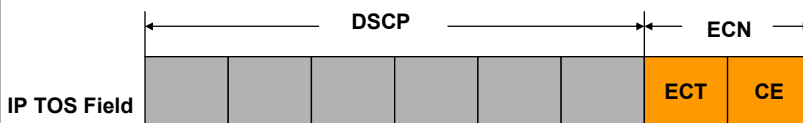


- RED performs "Active Queue Management" (AQM) and drops packets before congestion occurs
 - ◆ But an uncertainty remains whether congestion will occur at all
- RED is known as "difficult to tune"
 - ◆ Goal: Self-tuning RED
 - ◆ Running estimate weighted moving average (EWMA) of the average queue size

Explicit Congestion Notification (ECN)



- Traditional TCP stacks only use **packet loss** as indicator to reduce window size
 - ◆ But some applications are sensitive to packet loss and delays
- Routers with ECN enabled **mark packets** when the average queue depth exceeds a threshold
 - ◆ Instead of randomly dropping them
 - ◆ Hosts may reduce window size upon receiving ECN-marked packets
- Least significant two bits of IP TOS used for ECN



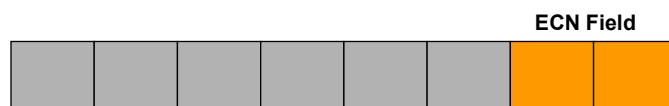
Obsolete (but widely used) RFC 2481 notation of these two bits:

ECT ECN-Capable Transport
 CE Congestion Experienced

Usage of CE and ECT



- RFC 3168 redefines the use of the two bits: ECN-supporting hosts should set one of the **two ECT code points**
 - ◆ ECT(0) or ECT(1)
 - ◆ ECT(0) SHOULD be preferred
- Routers that experience congestion set the CE code point in packets with ECT code point set (otherwise: RED)
- If average queue depth is exceeding max-threshold: Tail-drop
- If CE already set: forward packet normally (abuse!)

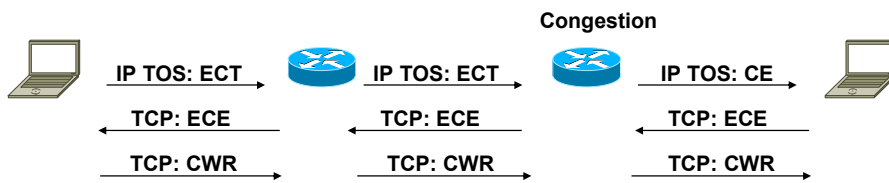


Non ECN-capable transport	0	0	
Codepoints for ECN-capable transport	0	1	ECT(1)
	1	0	ECT(0)
CE codepoint	1	1	

CWR and ECE



- RFC 3168 also introduced two new TCP flags
 - ♦ ECN Echo (ECE)
 - ♦ Congestion Window Reduced (CWR)
- Purpose:
 - ♦ ECE used by data receiver to inform the data sender when a CE packet has been received
 - ♦ CWR flag used by data sender to inform the data receiver that the congestion window has been reduced



Part of TCP header:



ECN Configuration



- Note: ECN is an extension to WRED
 - ♦ Therefore WRED must be enabled first !
- ECN will be applied on that traffic that is identified by WRED (e. g. dscp-based)

```
(config-pmap-c) # random-detect
(config-pmap-c) # random-detect ecn

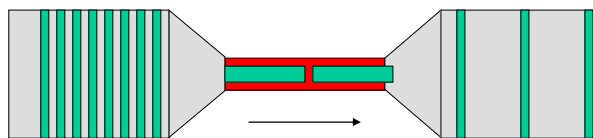
# show policy-map interface s0/1 !!! shows ECN setting
```

Note



- **CE is only set when average queue depth exceeds a threshold**
 - ◆ End-host would react immediately
 - ◆ Therefore ECN is not appropriate for short term bursts (similar as RED)
- **Therefore ECN is different as the related features in Frame Relay or ATM which acts also on short term (transient) congestion**

What happens when TCP window size is too Large?



- **TCP gets "ACK-clocking"**
 - ◆ Every node runs exactly at the bottleneck link rate
 - ◆ Queue depth has nothing to do with rate mismatch but is determined by the bottleneck and the reacting TCP stacks