

Congestion Control

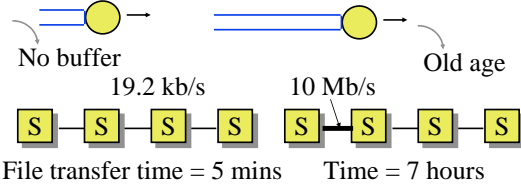
Arjan Duresi
Louisiana State University

Why Worry About Congestion?

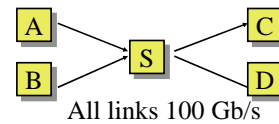
Q: Will the congestion problem be solved when:

- Memory becomes cheap (infinite memory)?
- Links become cheap (very high speed links)?
- Processors become cheap?

A: None of the above.



- How to deal with congestion?
- Queuing Discipline
- Reacting to Congestion
- Avoiding Congestion TCP
- RED, ECN, MECN, LED



Conclusions:

- Congestion is a dynamic problem. Static solutions are not sufficient
- Bandwidth explosion
⇒ More unbalanced networks
- Buffer shortage is a symptom not the cause.

Future

Year
1980



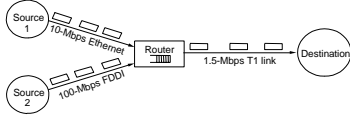
In 1990, the memory will be so cheap that you will not have to worry about paging, swapping, virtual memory, memory hierarchy, and....

Economic Reasons

- Network is a shared resource
Because it is expensive and needed occasionally (Like airplanes, emergency rooms)
- Most costs are fixed.
Cost for fiber, switches, laying fiber and maintaining them does not depend upon usage
⇒ Underutilization is expensive
- But overutilization leads to user dissatisfaction.
- Need a way to keep the network maximally utilized

Issues

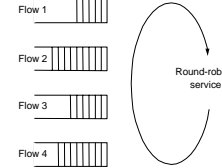
- ❑ Two sides of the same coin
 - ❑ pre-allocate resources so as to avoid congestion
 - ❑ control congestion if (and when) it occurs



- ❑ Two points of implementation
 - ❑ hosts at the edges of the network (transport protocol)
 - ❑ routers inside the network (queuing discipline)
- ❑ Underlying service model
 - ❑ best-effort (assume for now)
 - ❑ multiple *qualities of service* (later)

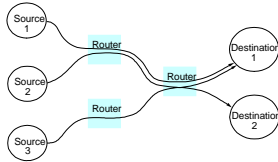
Queuing Discipline

- ❑ First-In-First-Out (FIFO)
 - ❑ does not discriminate between traffic sources
- ❑ Fair Queuing (FQ)
 - ❑ explicitly segregates traffic based on flows
 - ❑ ensures no flow captures more than its share of capacity
 - ❑ variation: weighted fair queuing (WFQ)
- ❑ Problem?



Framework

- ❑ Connectionless flows
 - ❑ sequence of packets sent between source/destination pair
 - ❑ maintain *soft state* at the routers



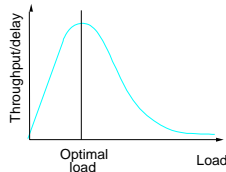
- ❑ Taxonomy
 - ❑ router-centric versus host-centric
 - ❑ reservation-based versus feedback-based
 - ❑ window-based versus rate-based

FQ Algorithm

- ❑ Suppose clock ticks each time a bit is transmitted
- ❑ Let P_i denote the length of packet i
- ❑ Let S_i denote the time when start to transmit packet i
- ❑ Let F_i denote the time when finish transmitting packet i
- ❑ $F_i = S_i + P_i$
- ❑ When does router start transmitting packet i ?
 - ❑ if before router finished packet $i - 1$ from this flow, then immediately after last bit of $i - 1$ (F_{i-1})
 - ❑ if no current packets for this flow, then start transmitting when arrives (call this A_i)
- ❑ Thus: $F_i = \text{MAX}(F_{i-1}, A_i) + P_i$

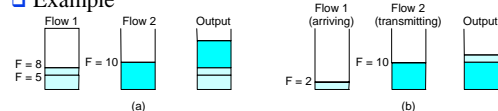
Evaluation

- ❑ Fairness
- ❑ Power (ratio of throughput to delay)



FQ Algorithm (cont)

- ❑ For multiple flows
 - ❑ calculate F_i for each packet that arrives on each flow
 - ❑ treat all F_i 's as timestamps
 - ❑ next packet to transmit is one with lowest timestamp
- ❑ Not perfect: can't preempt current packet
- ❑ Example

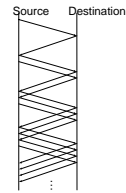


TCP Congestion Control

- Idea
 - assumes best-effort network (FIFO or FQ routers) each source determines network capacity for itself
 - uses implicit feedback
 - ACKs pace transmission (*self-clocking*)
- Challenge
 - determining the available capacity in the first place
 - adjusting to changes in the available capacity

AIMD (cont)

- Algorithm
 - increment **CongestionWindow** by one packet per RTT (*linear increase*)
 - divide **CongestionWindow** by two whenever a timeout occurs (*multiplicative decrease*)
- In practice: increment a little for each ACK
 - $\text{Increment} = (\text{MSS} * \text{MSS}) / \text{CongestionWindow}$
 - $\text{CongestionWindow} += \text{Increment}$

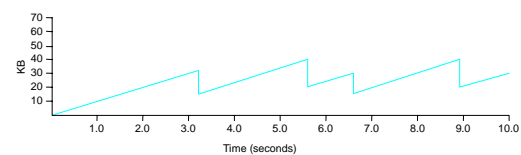


Additive Increase/Multiplicative Decrease

- Objective: adjust to changes in the available capacity
 - New state variable per connection: **CongestionWindow**
 - limits how much data source has in transit
- $$\text{MaxWin} = \text{MIN}(\text{CongestionWindow}, \text{AdvertisedWindow})$$
- $$\text{EffWin} = \text{MaxWin} - (\text{LastByteSent} - \text{LastByteAcked})$$
- Idea:
 - increase **CongestionWindow** when congestion goes down
 - decrease **CongestionWindow** when congestion goes up

AIMD (cont)

- Trace: sawtooth behavior

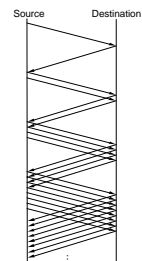


AIMD (cont)

- Question: how does the source determine whether or not the network is congested?
- Answer: a timeout occurs
 - timeout signals that a packet was lost
 - packets are seldom lost due to transmission error
 - lost packet implies congestion

Slow Start

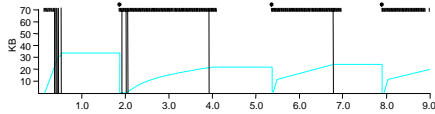
- Objective: determine the available capacity in the first
- Idea:
 - begin with **CongestionWindow** = 1 packet
 - double **CongestionWindow** each RTT (increment by 1 packet for each ACK)



Slow Start (cont)

- Exponential growth, but slower than all at once
- Used...
 - when first starting connection
 - when connection goes dead waiting for timeout

Trace



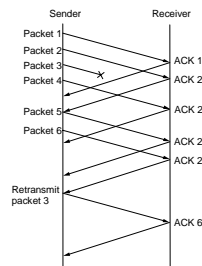
- Problem: lose up to half a **CongestionWindow**'s worth of data

Congestion Avoidance

- TCP's strategy
 - control congestion once it happens
 - repeatedly increase load in an effort to find the point at which congestion occurs, and then back off
- Alternative strategy
 - predict when congestion is about to happen
 - reduce rate before packets start being discarded
 - call this congestion *avoidance*, instead of congestion *control*
- Two possibilities
 - router-centric: DECbit and RED Gateways
 - host-centric: TCP Vegas

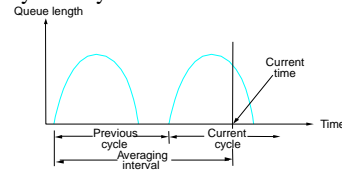
Fast Retransmit and Fast Recovery

- Problem: coarse-grain TCP timeouts lead to idle periods
- Fast retransmit: use duplicate ACKs to trigger retransmission



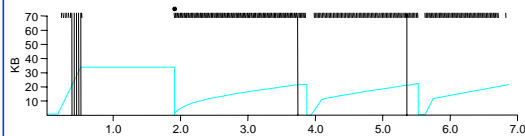
DECbit

- Add binary congestion bit to each packet header
- Router
 - monitors average queue length over last busy+idle cycle



- set congestion bit if average queue length > 1
- attempts to balance throughput against delay

Results



- Fast recovery
 - skip the slow start phase
 - go directly to half the last successful **CongestionWindow** (**ssthresh**)

End Hosts

- Destination echoes bit back to source
- Source records how many packets resulted in set bit
- If less than 50% of last window's worth had bit set
 - increase **CongestionWindow** by 1 packet
- If 50% or more of last window's worth had bit set
 - decrease **CongestionWindow** by 0.875 times

Random Early Detection (RED)

- Notification is implicit
 - just drop the packet (TCP will timeout)
 - could make explicit by marking the packet
- Early random drop
 - rather than wait for queue to become full, drop each arriving packet with some *drop probability* whenever the queue length exceeds some *drop level*

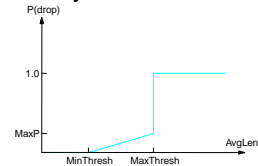
RED Details (cont)

- Computing probability P

$$\text{TempP} = \text{MaxP} * (\text{AvgLen} - \text{MinThreshold}) / (\text{MaxThreshold} - \text{MinThreshold})$$

$$P = \text{TempP} / (1 - \text{count} * \text{TempP})$$

- Drop Probability Curve



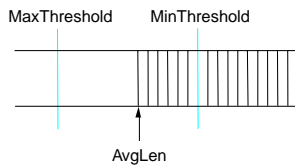
RED Details

- Compute average queue length

$$\text{AvgLen} = (1 - \text{Weight}) * \text{AvgLen} + \text{Weight} * \text{SampleLen}$$

$0 < \text{Weight} < 1$ (usually 0.002)

SampleLen is queue length each time a packet arrives



Tuning RED

- Probability of dropping a particular flow's packet(s) is roughly proportional to the share of the bandwidth that flow is currently getting
- **MaxP** is typically set to 0.02, meaning that when the average queue size is halfway between the two thresholds, the gateway drops roughly one out of 50 packets.
- If traffic is bursty, then **MinThreshold** should be sufficiently large to allow link utilization to be maintained at an acceptably high level
- Difference between two thresholds should be larger than the typical increase in the calculated average queue length in one RTT; setting **MaxThreshold** to twice **MinThreshold** is reasonable for traffic on today's Internet
- Penalty Box for Offenders

RED Details (cont)

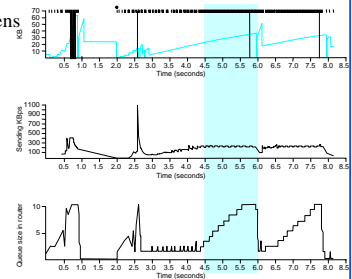
- Two queue length thresholds


```

if AvgLen <= MinThreshold then
    enqueue the packet
if MinThreshold < AvgLen < MaxThreshold
then
    calculate probability P
    drop arriving packet with probability P
if MaxThreshold <= AvgLen then
    drop arriving packet
            
```

TCP Vegas

- Idea: source watches for some sign that router's queue is building up and congestion will happen too; e.g.,
 - RTT grows
 - sending rate flattens



Algorithm

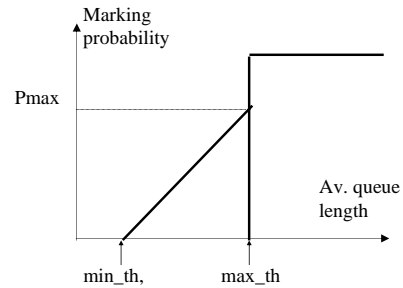
- Let **BaseRTT** be the minimum of all measured RTTs (commonly the RTT of the first packet)
- If not overflowing the connection, then

$$\text{ExpectRate} = \text{CongestionWindow} / \text{BaseRTT}$$
- Source calculates sending rate (**ActualRate**) once per RTT
- Source compares **ActualRate** with **ExpectRate**

```

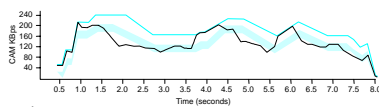
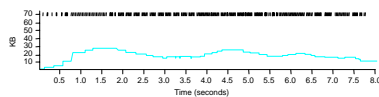
Diff = ExpectedRate - ActualRate
if Diff <  $\alpha$ 
    increase CongestionWindow linearly
else if Diff >  $\beta$ 
    decrease CongestionWindow linearly
else
    leave CongestionWindow unchanged
    
```

ECN - Router Marking



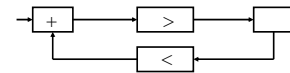
Algorithm (cont)

- Parameters
 - $\alpha = 1$ packet
 - $\beta = 3$ packets



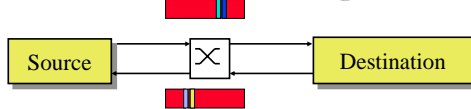
- Even faster retransmit
 - keep fine-grained timestamps for each packet
 - check for timeout on first duplicate ACK

Multilevel ECN



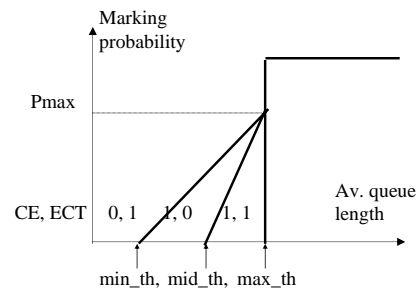
- ECN - Feedback Control Mechanism
- More Feedback Information => Enables more appropriate response from the Source
- Better Congestion Control:
 - More Effective use of network resources
 - Less Losses
 - Less Delay
- Our MECN paper won "best paper" award in the 5th World Multiconference on Systemics, Cybernetics and Informatics SCT'2001
- Use Control Theory to set parameters in RED, ECN and MECN

Buffer Management using ECN



- Explicit Congestion Notification
- RFC 2481, expected to be standardized and widely implemented soon.
- Two bits in IP header: Congestion Experienced, ECN Capable Transport: bits 6 and 7 in the TOS octet in IPv4, or the Traffic Class octet in IPv6
- Two bits in TCP header: ECN Echo, Congestion Window Reduced

MECN - Router Marking



TCP Source Response

Congestion State	cwnd change
No congestion	Increase 'cwnd' additively
Incipient congestion	Decrease multiplicatively by β_1
Moderate congestion	Decrease multiplicatively by β_2
Severe congestion	Decrease multiplicatively by β_3

Performance vs. Stability

□ Sensitivity – How fast the system reacts

□ Higher Sensitivity – less jitter

$$Sensitivity = \frac{I + K_{MECN}}{I + K_0}$$

□ Delay Margin – Stability of the system

□ Higher Delay Margin – more throughput

$$DM(w_g) \approx \frac{\pi \tan^{-1}(\frac{w_g}{K})}{w_g} - R_0$$

where: $w_g = K\sqrt{K_{MECN}^2 - 1}$, $K = -C \ln(1-\alpha)$, α - is the queue averaging weight

Control Theoretic Analysis

□ In RED, ECN, setting parameters is difficult

□ People “rules of thumb”

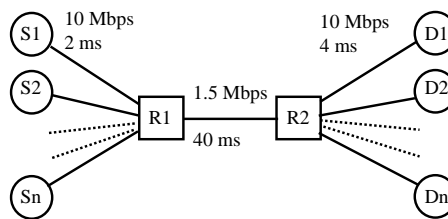
□ In MECN – more parameters

□ To resolve this problem:

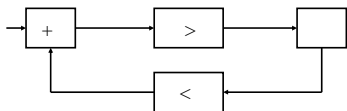
□ We have modeled RED, ECN and MECN using control theory

□ Used new metrics to optimize parameters

Simulation Configuration



Modeling MECN



$$\dot{W}(t) = \frac{1}{R(t)} - \frac{W(t)W(t-R(t))}{\beta_1 R(t-R(t))} - P_1(t-R(t)) - \frac{W(t)W(t-R(t))}{\beta_2 R(t-R(t))} - P_2(t-R(t))$$

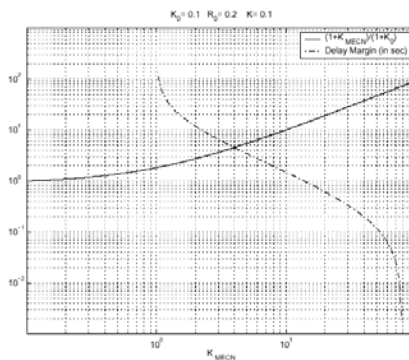
$$\dot{q}(t) = \left[\frac{N(t)}{R(t)} W(t) - C \right]^+$$

$$K_{MECN} = \frac{(R_0 C)^3}{2N^2} \left(\frac{L_{RED_1}}{\beta_1} + \frac{L_{RED_2}}{\beta_2} \right)$$

$$L_{RED_1} = \frac{P_{max}}{\max_{th} - mid_{th}}$$

$$L_{RED_2} = \frac{P_{max}}{\max_{th} - mid_{th}}$$

Performance vs. Stability



MECN performance

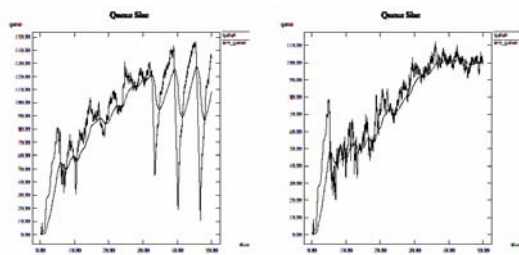


Figure 11. Effect of parameter tuning on MECN: low K_{MECN}

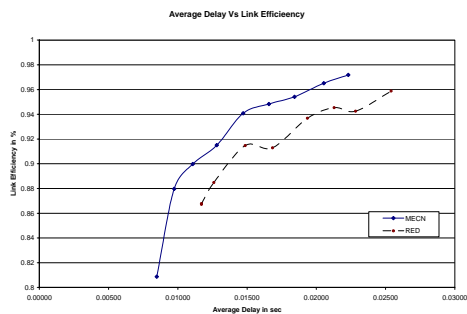
Jitter = 30 ms

Figure 12. Effect of parameter tuning on MECN: high K_{MECN}

Jitter = 15 ms

Adaptive MECN

- Change the parameters automatically depending on the traffic



MECN performance

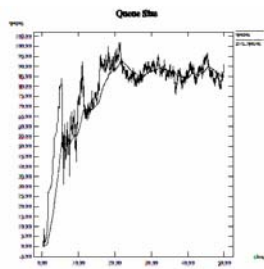
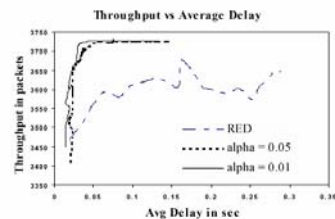


Figure 13. Effect of parameter tuning on MECN: higher K_{MECN}

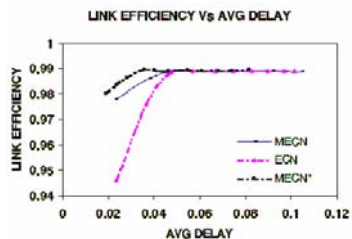
Jitter = 7 ms

Load Early Detection - LED

- Network load a better a better metric to measure congestion.
Network load = (bytes received)/(bytes processed)
- We have proved theoretically that LED outperforms RED



MECN performance



- Our analysis methodology is a complete framework to compare and optimize congestion control schemes

Summary



- Congestion Management
- Queuing Discipline
- Reacting to Congestion
- Avoiding Congestion TCP
- RED, ECN, MECN, LED