The UNIVERSITY *of* NORTH CAROLINA
at CHAPEL HILL

Making AQM Work: An Efficient Alternative to ECN

Long Le, Jay Aikat, Kevin Jeffay, and Don Smith

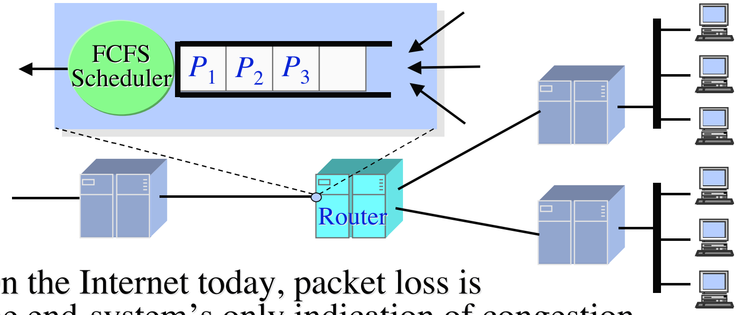
October 2003

http://www.cs.unc.edu/Research/dirt

Making AQM Work
Outline

- Background: Router-based congestion control
  - Active Queue Management
  - Explicit Congestion Notification
- State of the art in active queue management (AQM)
  - Control theoretic *v*. traditional randomized dropping AQM
- Do AQM schemes work?
  - An empirical study of the effect of AQM on web performance
- Analysis of AQM performance
  - The case for *differential congestion notification* (DCN)
- A DCN prototype and its empirical evaluation

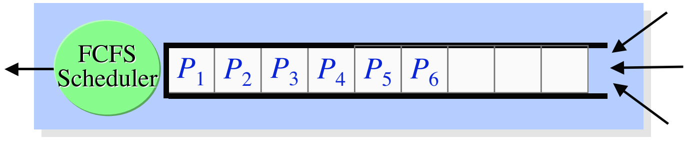
Router-Based Congestion Control
Status quo



- On the Internet today, packet loss is the end-system’s only indication of congestion
- As switch’s queues overflow, arriving packets are dropped
  - “Drop-tail” FIFO queuing is the default
- TCP end-systems detect loss and respond by reducing their transmission rate

---

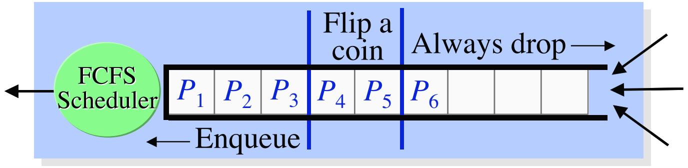
Router-Based Congestion Control
The case against drop-tail queuing



- Large (full) queues in routers are a bad thing
  - End-to-end latency is dominated by the length of queues at switches in the network
- Allowing queues to overflow is a bad thing
  - Connections that transmit at high rates can starve connections that transmit at low rates
  - Causes connections to synchronize their response to congestion and become unnecessarily bursty

Router-Based Congestion Control

Active queue management (AQM)

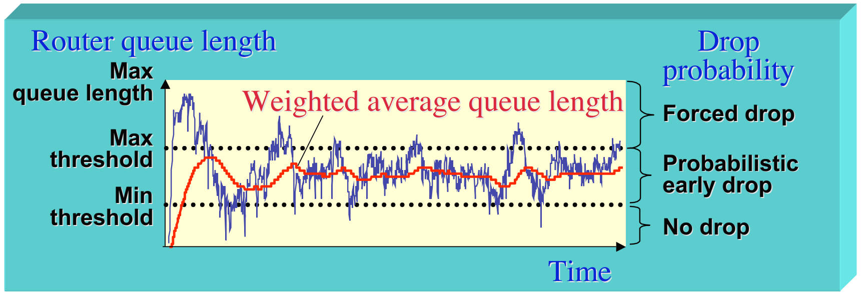


- Key concept: Drop packets *before* a queue overflows to signal *incipient* congestion to end-systems
- Basic mechanism: When the queue length exceeds a threshold, packets are probabilistically dropped
- *Random Early Detection* (RED) AQM:
  - Always enqueue if queue length less than a low-water mark
  - Always drop if queue length is greater than a high-water mark
  - Probalistically drop/enqueue if queue length is in between

---

Active Queue Management

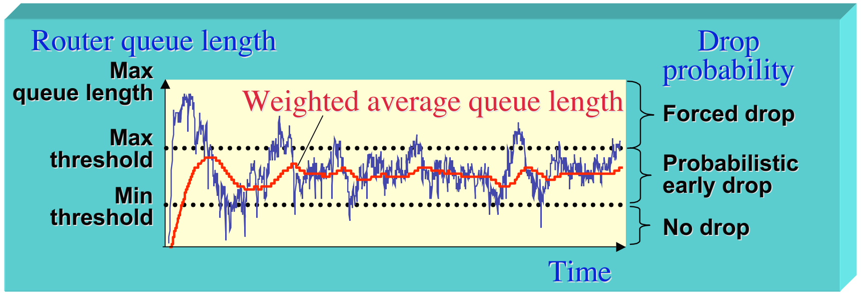
The RED Algorithm [Floyd & Jacobson 93]

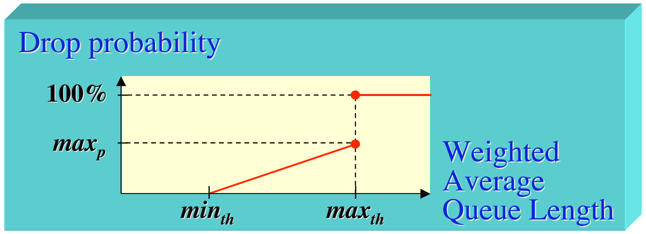


- RED computes a weighted moving average of queue length to accommodate bursty arrivals
- Drop probability is a function of the current average queue length
  - The larger the queue, the higher the drop probability

Active Queue Management

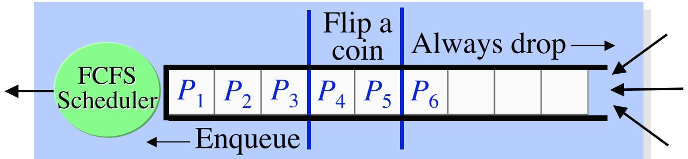
The RED Algorithm [Floyd & Jacobson 93]





Active Queue Management

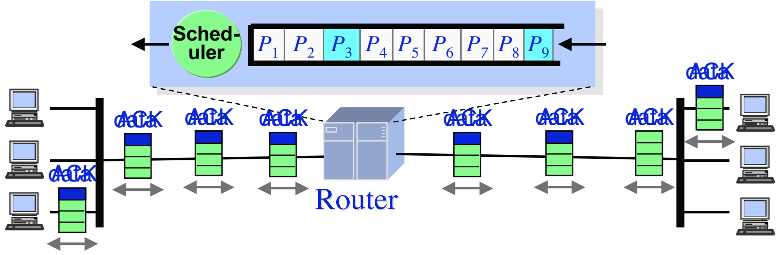
Explicit Congestion Notification (ECN)



- Dropping packets is a simple means of signaling congestion but it's less than ideal
  - It may take a long time for a sender to detect and react to congestion signaled by packet drops
  - There are subtle fairness issues in the way flows are treated
- ECN: Instead of dropping packets, send an explicit signal back to the sender to indicate congestion
  - (An old concept: ICMP Source Quench, DECbit, ATM, …)

Explicit Congestion Notification

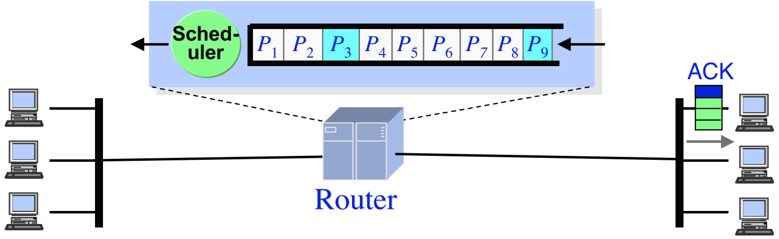
Overview


Sched-uler
P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9
ACK
data
Router

- Modify a RED router to “mark” packets rather than dropping them
- Set a bit in a packet’s header and forward towards the ultimate destination
- A receiver recognizes the marked packet and sets a corresponding bit in the next outgoing ACK

Explicit Congestion Notification

Overview

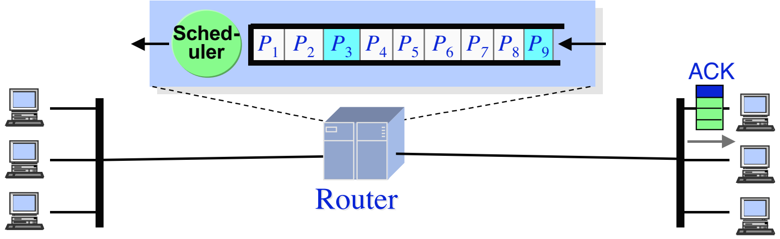

Sched-uler
P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9
ACK
Router

- When a sender receives an ACK with ECN it invokes a response similar to that for packet loss:
  - Halve the congestion window *cwnd* and halve the slow-start threshold *ssthresh*
  - Continue to use ACK-clocking to pace transmission of data packets

---

Explicit Congestion Notification

Overview

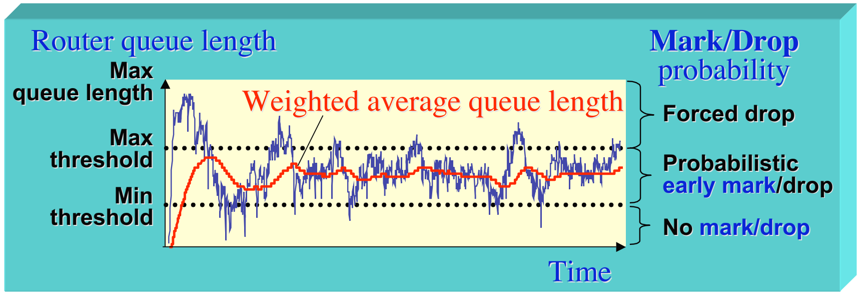

Sched-uler
P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9
ACK
Router

- When a sender receives an ACK with ECN it invokes a response similar to that for packet loss
- In any given RTT, a sender should react to either ECN or packet loss *but not both*!
  - Once a response has begun, wait until all outstanding data has been ACKed before beginning a new response

---

Explicit Congestion Notification

Putting the pieces together: AQM + ECN

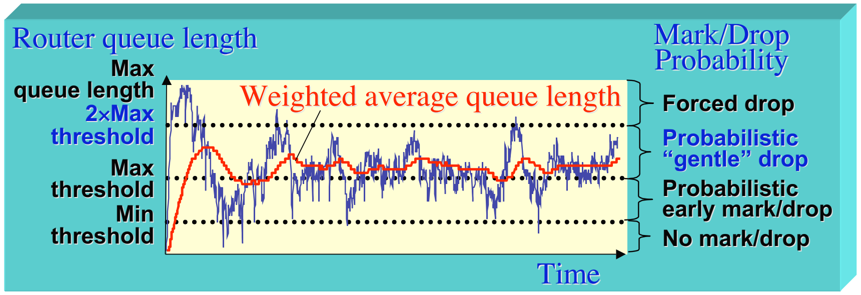


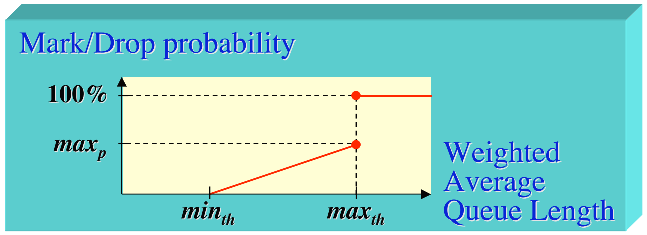
- If a RED router detects congestion it will mark arriving packets
- The router will then forward marked packets from ECN-capable senders…
- …and drop marked packets from all other senders

Making AQM Work
Outline

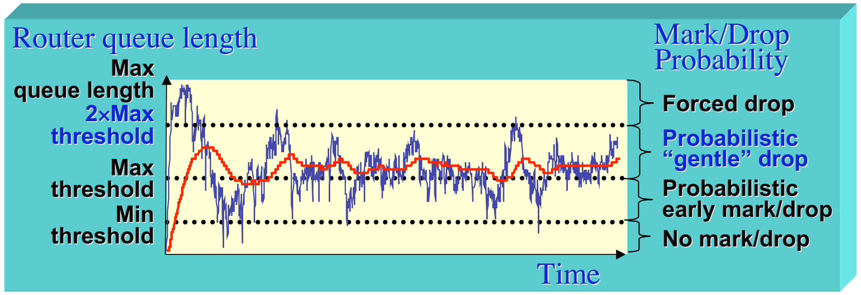
- Background: Router-based congestion control
  - Active Queue Management
  - Explicit Congestion Notification
- State of the art in active queue management (AQM)
  - Control theoretic *v.* traditional randomized dropping AQM
- Do AQM schemes work?
  - An empirical study of the effect of AQM on web performance
- Analysis of AQM performance
  - The case for *differential congestion notification* (DCN)
- A DCN prototype and its empirical evaluation

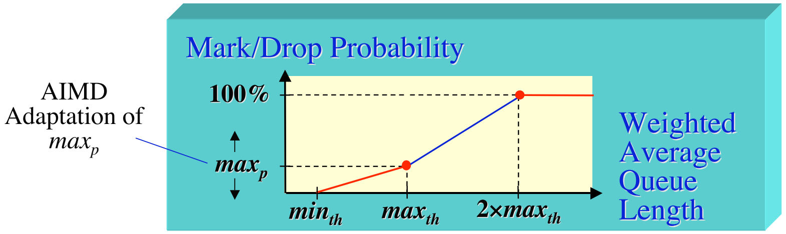
The State of the ART in AQM
Adaptive/Gentle RED (ARED)



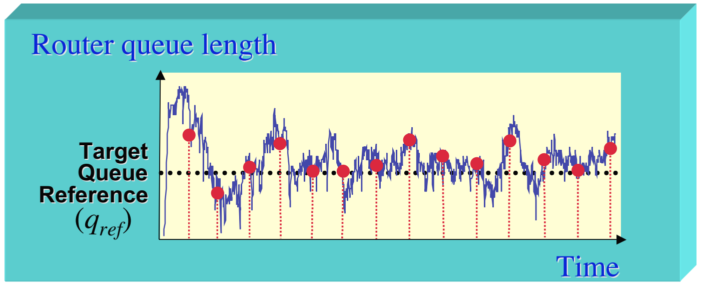


The State of the ART in AQM
Adaptive/Gentle RED (ARED)



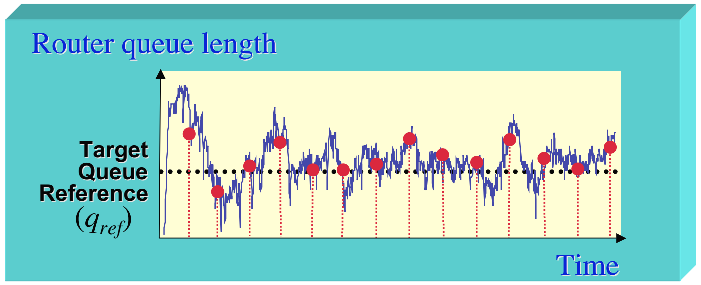


The State of the ART in AQM
The Proportional Integral (PI) controller



- PI attempts to maintain an explicit target queue length
- PI samples instantaneous queue length at fixed intervals and computes a mark/drop probability at k^{th} sample:
  - $p(kT) = a \times (q(kT) - q_{ref}) - b \times (q((k-1)T) - q_{ref}) + p((k-1)T)$
  - a, b, and T depend on link capacity, maximum RTT and the number of flows at a router

The State of the ART in AQM
Random Exponential Marking (REM)



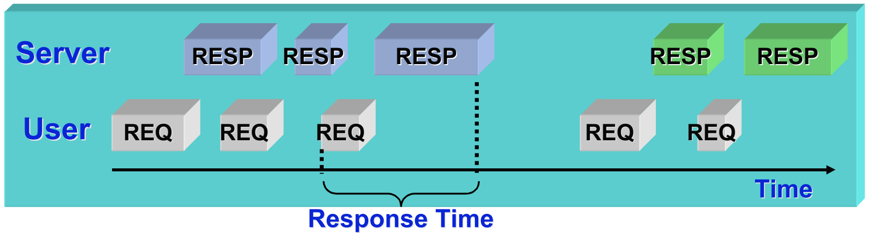
- REM is similar to PI (though differs in details)
- REM mark/drop probability depends on:
  - Difference between input and output rate
  - Difference between instantaneous queue length and target
  - $p(t) = p(t–1) + \gamma\,[\alpha\,(q(t) – q_{ref})) + x(t) – c]$
  - $prob(t) = 1 – \phi^{-p(t)}$, $\phi > 1$ a constant

Do AQM Schemes Work?
Evaluation methodology

- Evaluate AQM schemes through “live simulation”
- Emulate the browsing behavior of a large population of users surfing the web in a laboratory testbed
  - Construct a physical network emulating a congested peering link between two ISPs
  - Generate synthetic HTTP requests and responses but transmit over real TCP/IP stacks, network links, and switches

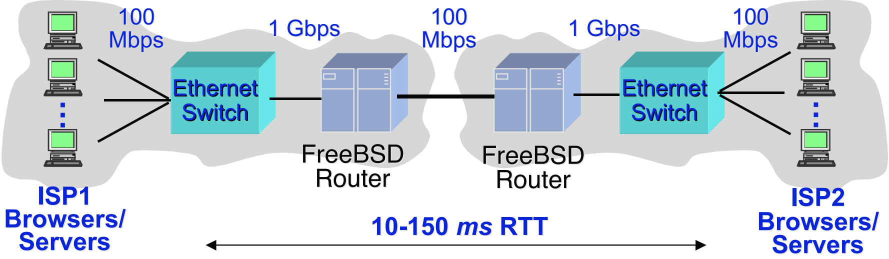
---

Experimental Methodology
HTTP traffic generation


Server
RESP
RESP
RESP
RESP
RESP
User
REQ
REQ
REQ
REQ
REQ
Time
Response Time

- Synthetic web traffic generated using the UNC HTTP model [SIGMETRICS 2001, MASCOTS 2003]
- Primary random variables:
  - Request sizes/Reply sizes
  - User think time
  - Persistent connection usage
  - Nbr of objects per persistent connection
  - Number of embedded images/page
  - Number of parallel connections
  - Consecutive documents per server
  - Number of servers per page

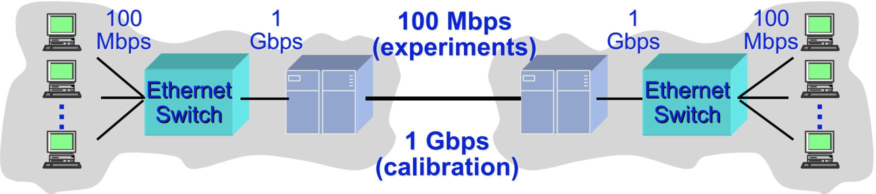
Experimental Methodology
Testbed emulating an ISP peering link


100 Mbps
1 Gbps
100 Mbps
1 Gbps
100 Mbps
Ethernet Switch
FreeBSD Router
FreeBSD Router
Ethernet Switch
ISP1 Browsers/ Servers
10-150 *ms* RTT
ISP2 Browsers/ Servers

- AQM schemes implemented in FreeBSD routers using ALTQ kernel extensions
- End-systems either a traffic generation client or server
  - Use *dummynet* to provide *per-flow* propagation delays
  - Two-way traffic generated, equal load generated in each direction

Experimental Methodology

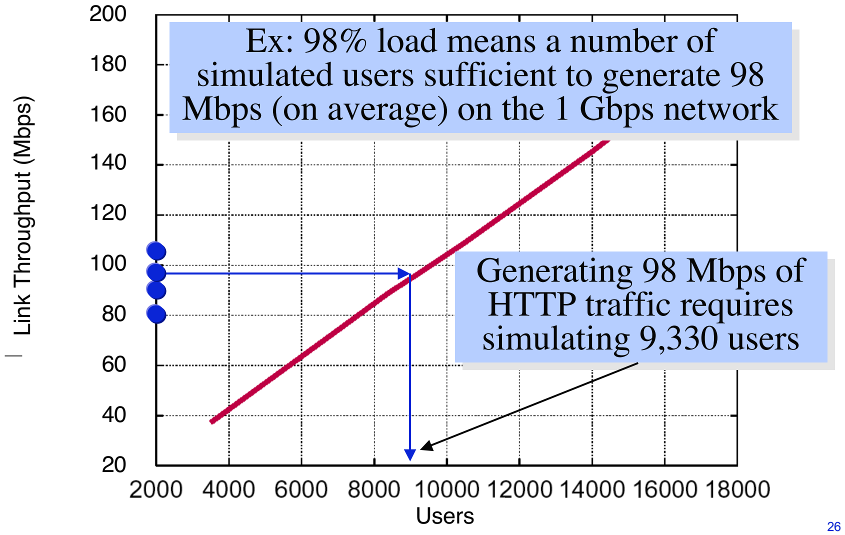
1 Gbps network calibration experiments



- Experiments run on a congested 100 Mbps link
- Primary simulation parameter: Number of simulated browsing users
- Run calibration experiments on an uncongested 1 Gbps link to relate simulated user populations to average link utilization
    - (And to ensure offered load is linear in the number of simulated users — *i.e.*, that end-systems are not a bottleneck)

Experimental Methodology

1 Gbps network calibration experiments



Ex: 98% load means a number of simulated users sufficient to generate 98 Mbps (on average) on the 1 Gbps network

Generating 98 Mbps of HTTP traffic requires simulating 9,330 users

---

Experimental Methodology

Experimental plan

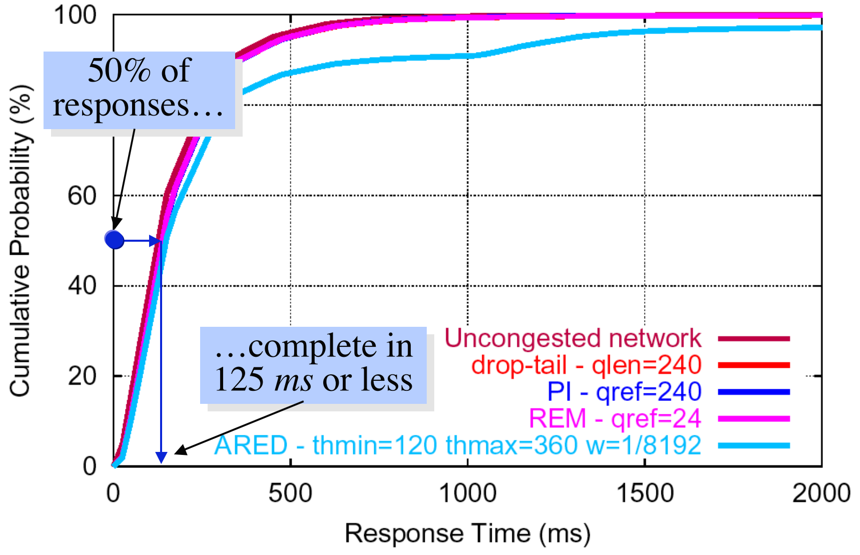
	80% 90% 98% 105%
uncongested drop-tail ARED PI REM	*loss rate* *utilization* *response times* *completed requests*

- Run experiments with ARED, PI, and REM using their recommended parameter settings at different offered loads
- Compare results with drop-tail FIFO at the same offered loads…
    - (the "negative" baselines — the performance to beat)

  …and compare with performance on the 1 Gbps network
    - (the "positive" baseline — the performance to achieve)
- Redo the experiments with ECN

Experimental Results — 80% Load

Performance with packet drops

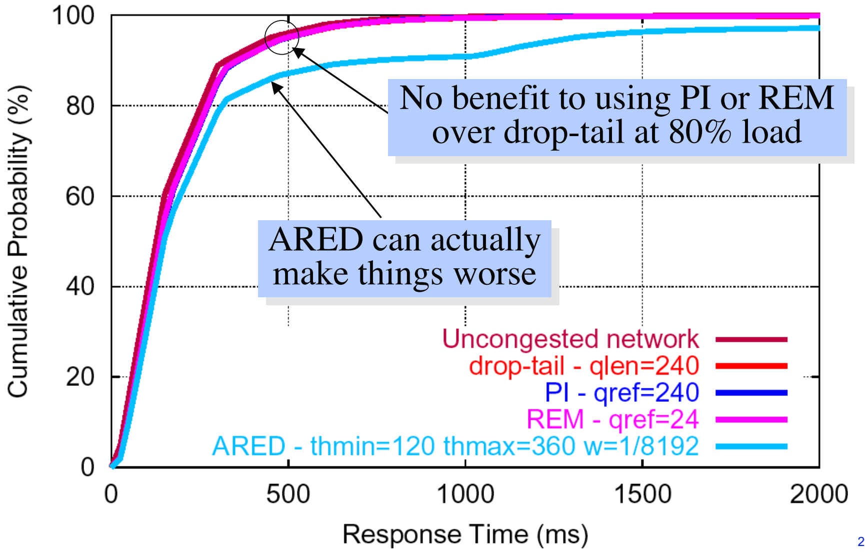


50% of responses…

…complete in 125 *ms* or less

Experimental Results — 80% Load

Performance with packet drops

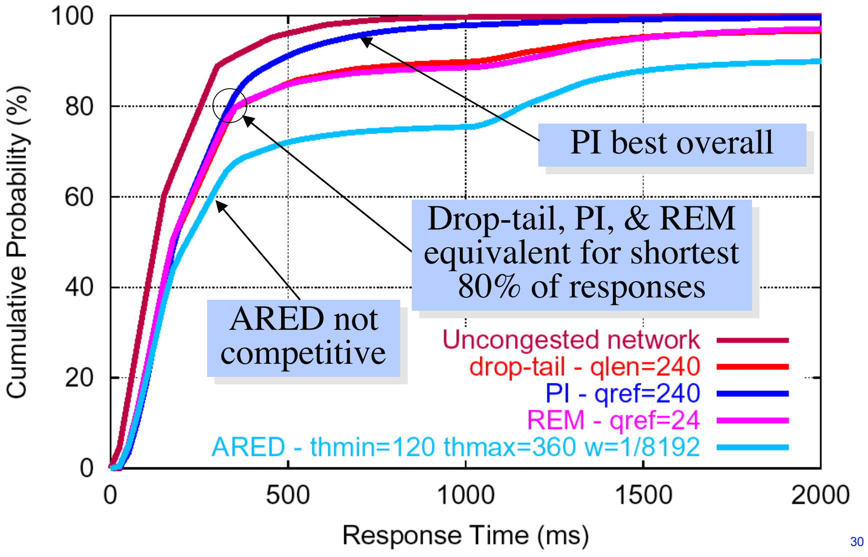


No benefit to using PI or REM over drop-tail at 80% load

ARED can actually make things worse

Experimental Results — 90% Load

Performance with packet drops



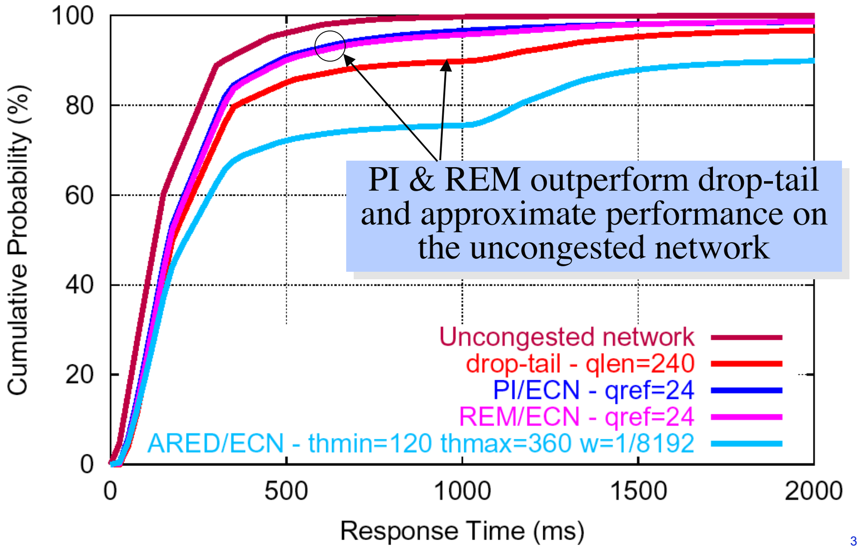
PI best overall

Drop-tail, PI, & REM equivalent for shortest 80% of responses

ARED not competitive

ECN Results — 90% Load

Comparison of all schemes

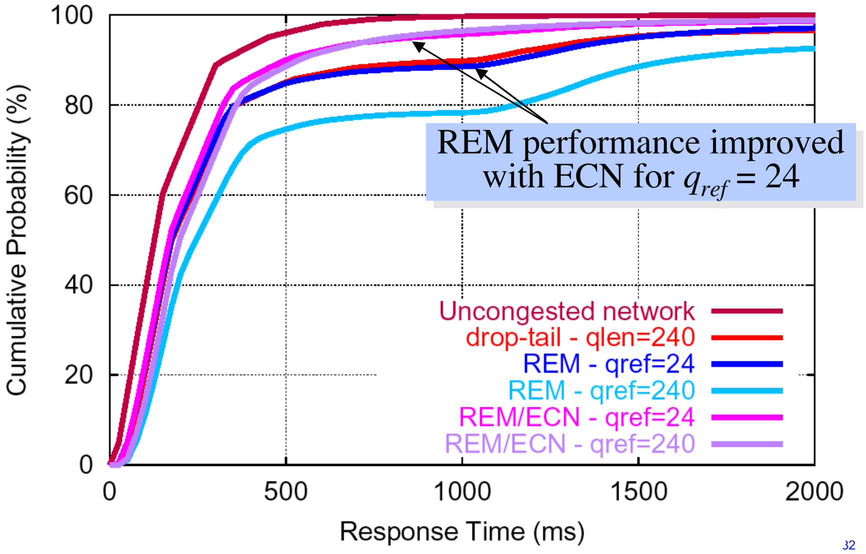


PI & REM outperform drop-tail and approximate performance on the uncongested network

---

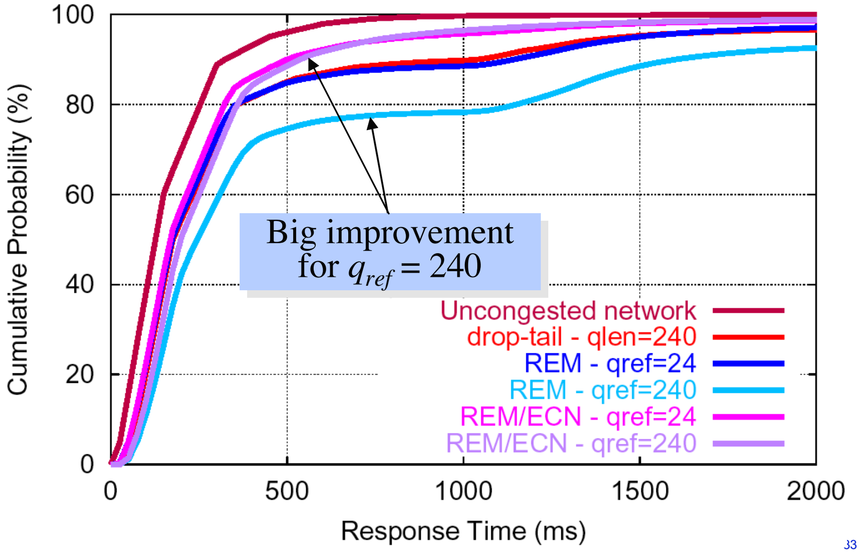
Impact of ECN on REM

Performance with/without ECN at 90% load



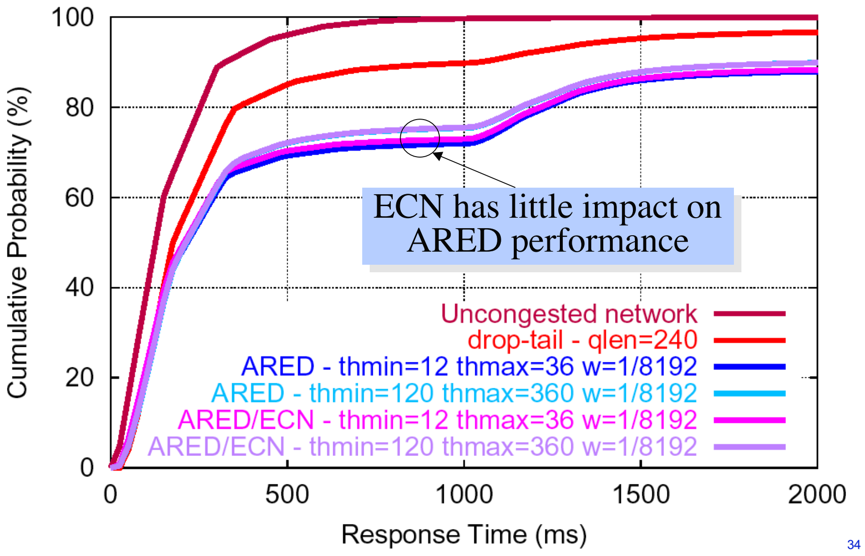
REM performance improved with ECN for $q_{ref} = 24$

Impact of ECN on REM
Performance with/without ECN at 90% load



Big improvement for $q_{ref} = 240$

Impact of ECN on ARED
Performance with/without ECN at 90% load

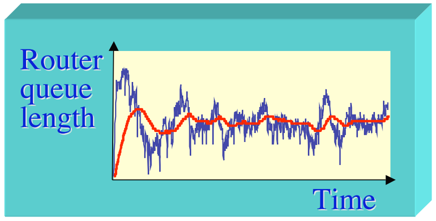


ECN has little impact on ARED performance

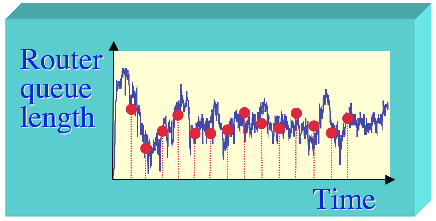
Do AQM Schemes Work?
Summary

- For offered loads up to 80% of link capacity, no AQM scheme gives better performance than drop-tail FIFO
  - All give comparable response time performance, loss rates, and link utilization
- For offered loads of 90% or greater…
  - Without ECN, PI results in a modest performance improvement over drop-tail and other AQM schemes
  - With ECN, both PI and REM provide significant performance improvement over drop-tail
- ARED consistently results in the poorest performance
  - Often worse than drop-tail FIFO

Discussion
Why does ARED perform so poorly?



Weighted Queue Length (RED)



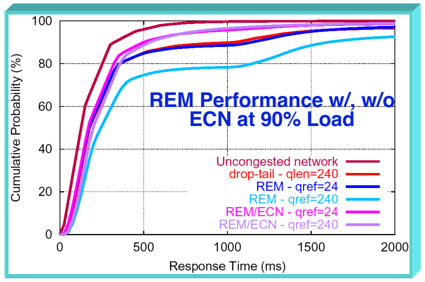
Instantaneous Queue Length (PI/REM)

- ARED bases mark/drop probability on the (weighted) average queue length
- PI, REM use instantaneous measures of queue length
- ARED’s reliance on the average queue length limits its ability to react effectively in the face of bursty traffic

Discussion

Why does ECN improve REM more than PI?

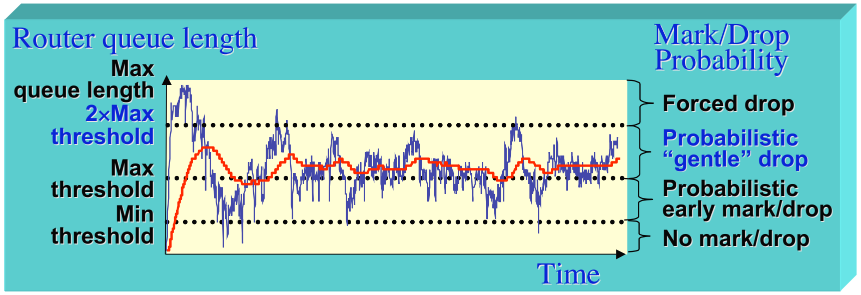
- Without ECN, REM drops more packets than PI
- REM causes more flows to experience multiple losses within a congestion window
  - Loss recovered through timeout rather than fast recovery



- In general ECN allows more flows to avoid timeouts
  - Thus ECN is ameliorating a design flaw in REM

Discussion

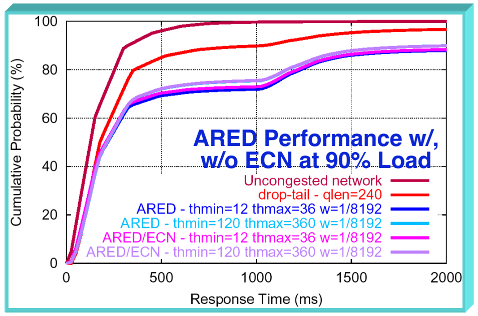
Why does ARED not benefit from ECN?


Router queue length
Max queue length
2×Max threshold
Max threshold
Min threshold
Time
Mark/Drop Probability
Forced drop
Probabilistic "gentle" drop
Probabilistic early mark/drop
No mark/drop

- ARED drops marked packets when average queue size is above max_{th}
- This is done to deal with potentially non-responsive flows
- We believe this policy is a premature optimization

Discussion

Why does ARED perform so poorly?



- PI and REM measure queue length in bytes
- By default RED measures in packets
  - But ARED does have a "byte mode"
- Drop/Mark probability in PI/REM biased by packet size
  - SYNs and pure ACKs have a lower drop probability in PI/REM
- Differentiating at the packet level is critical
  - Is it enough?

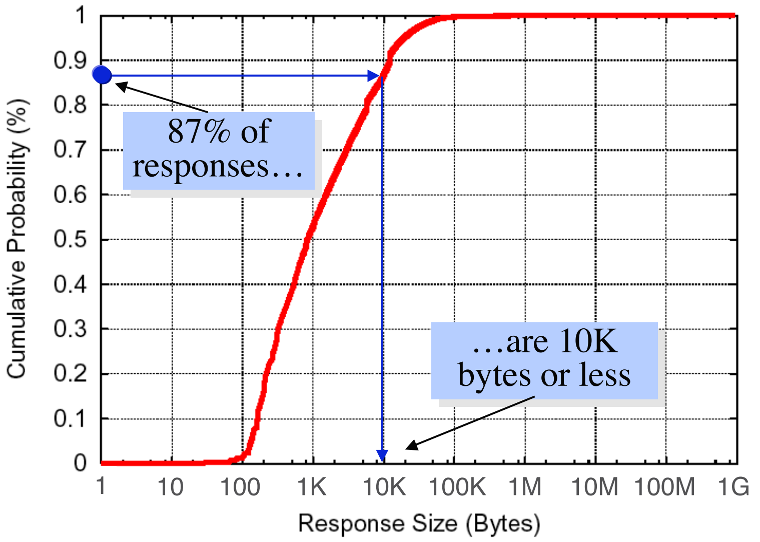
Discussion

Do AQM designs inherently require ECN?

- Claim: Differentiating between flows at the flow-level is important
- ECN is required for good AQM performance because it eliminates the need for short flows to retransmit (a significant fraction of their) data
  - With ECN, short flows (mostly) no longer retransmit data
  - But their performance is still hurt by AQM
- Why signal short flows at all?
  - They have no real transmission rate to adapt
  - Hence signaling these flows provides no benefit to the network and only hurts end-system performance

The Structure of Web Traffic

Distribution of response sizes

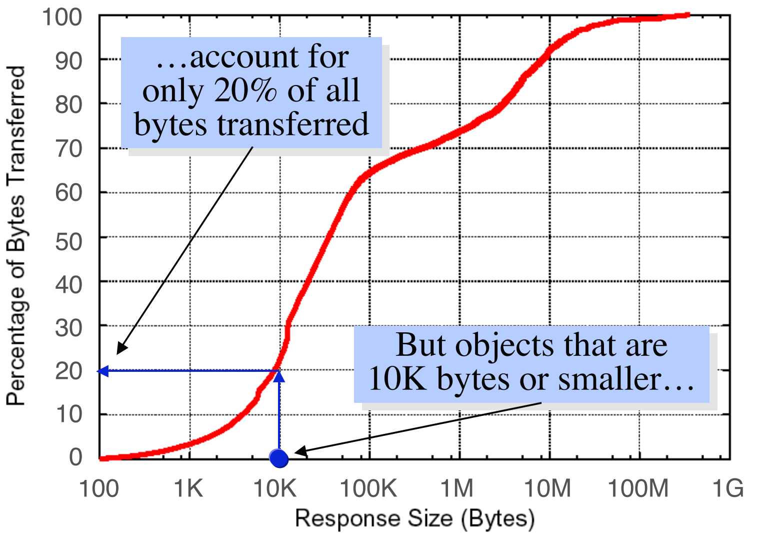


87% of responses…

…are 10K bytes or less

The Structure of Web Traffic

Percent of bytes transferred by response sizes



…account for only 20% of all bytes transferred

But objects that are 10K bytes or smaller…

Making AQM Work

Overview

- Background: Router-based congestion control
  - Active Queue Management
  - Explicit Congestion Notification
- State of the art in active queue management (AQM)
  - Control theoretic *v.* traditional randomized dropping AQM
- Do AQM schemes work?
  - An empirical study of the effect of AQM on web performance
- Analysis of AQM performance
  - The case for *differential congestion notification* (DCN)
- A DCN prototype and its empirical evaluation

Realizing Differential Notification

Issues and approach

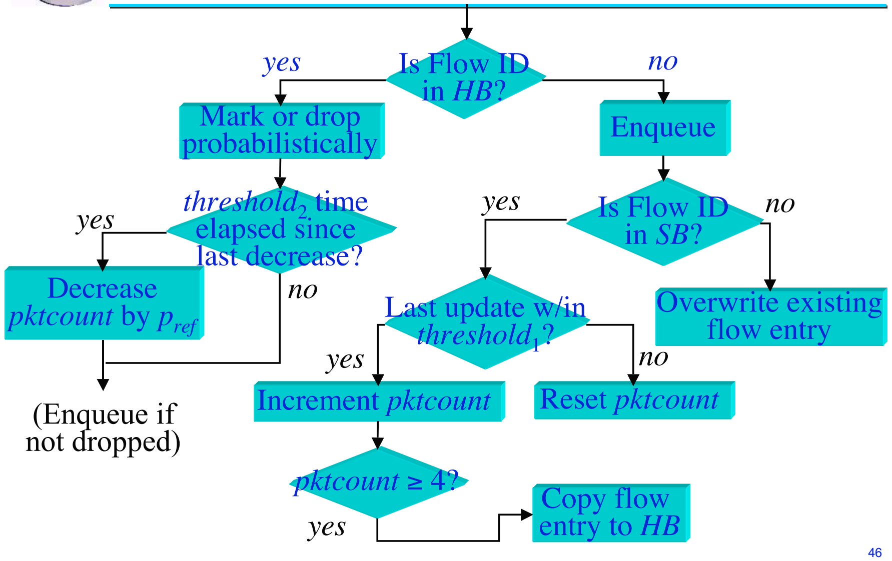
- How to identify packets belonging to long-lived, high bandwidth flows with minimal state?
  - Adopt the Estan, Varghese flow filtering scheme developed for traffic accounting [SIGCOMM 2002]
- How to determine when to signal congestion (by dropping packets)?
  - Use a PI-like scheme
- Differential treatment of flows an old idea:
  - FRED
  - SRED
  - CHOKe
  - SFB
  - AFD
  - RED-PD
  - RIO-PS
  - …

Classifying Flows
A score-boarding approach

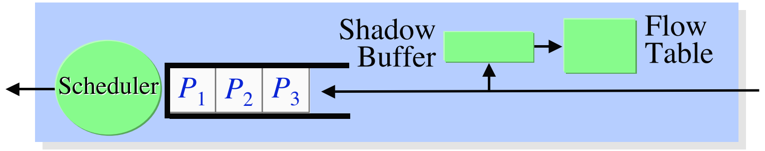
- Use two hash tables:
  - A "suspect" flow table HB ("high-bandwidth") and
  - A per-flow packet count table SB ("scoreboard")
  - Hash on IP addressing 4-tuple plus protocol number
- Arriving packets from flows in HB are subject to dropping
- Arriving packets from other flows are inserted into SB and tested to determine if the flow should be considered high-bandwidth
  - Use a simple packet count threshold for this determination

---

Classifying Flows
A score-boarding approach


Is Flow ID in *HB*?
yes
no
Mark or drop probabilistically
Enqueue
threshold$_2$ time elapsed since last decrease?
yes
no
Decrease *pktcount* by p_{ref}
(Enqueue if not dropped)
Is Flow ID in *SB*?
yes
no
Overwrite existing flow entry
Last update w/in *threshold*$_1$?
yes
no
Increment *pktcount*
Reset *pktcount*
pktcount ≥ 4?
yes
Copy flow entry to *HB*

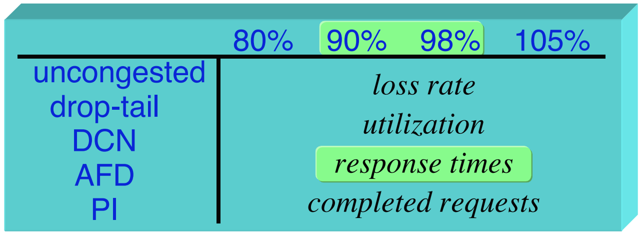
An Alternate Approach
AFD [Pan *et al.* 2003]


Shadow Buffer
Flow Table
Scheduler
P_1
P_2
P_3

"Approximate Fairness through Differential Dropping"

- Sample 1 out of every *s* packets and store in a *shadow buffer* of size *b*
- Estimate flow's rate as $r_{est} = R \frac{\#\ matches}{b}$
- Drop packet with probability $p = 1 - \frac{r_{fair}}{r_{est}}$

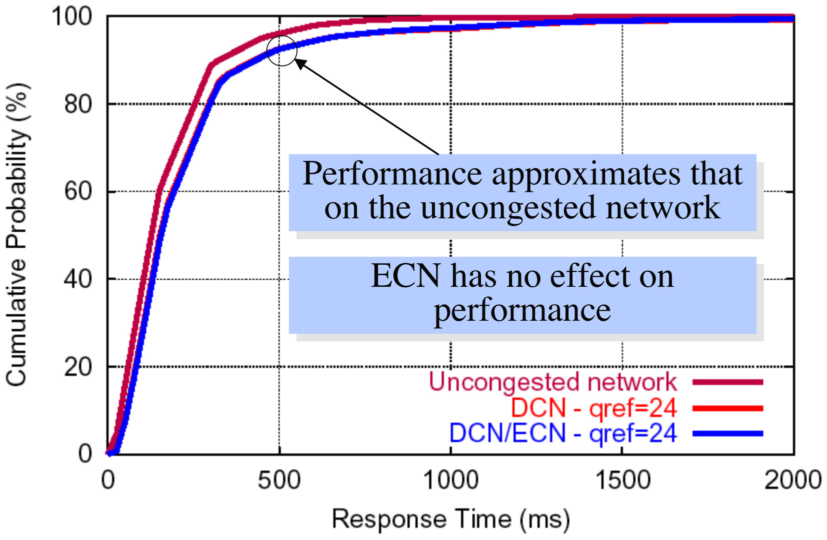
DCN Evaluation
Experimental plan


80%
90%
98%
105%
uncongested
drop-tail
DCN
AFD
PI
loss rate
utilization
response times
completed requests

- Run experiments with DCN, AFD, and PI at same offered loads as before
  - PI always uses ECN, test AFD with and without ECN
  - DCN always signals congestion via drops
- Compare DCN results against…
  - The better of PI or AFD (the performance to beat)
  - The uncongested network (the performance to approximate)

Experimental Results — 90% Load

DCN performance

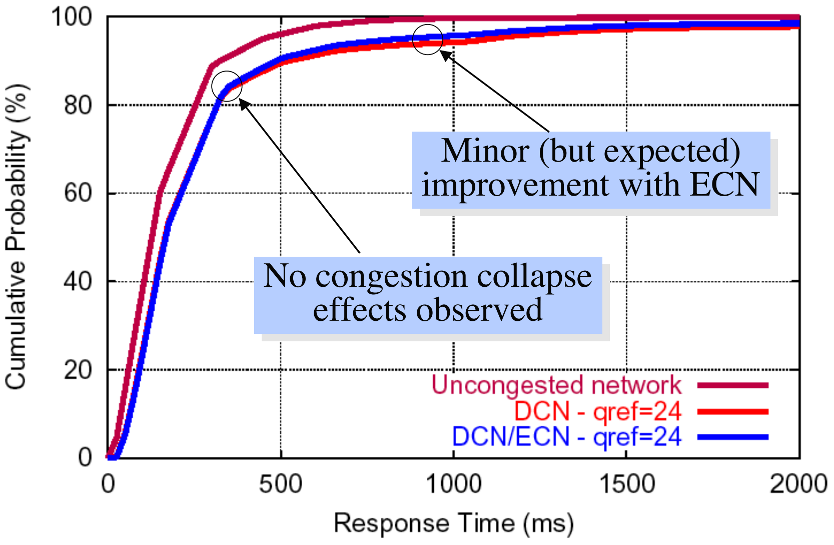


Performance approximates that on the uncongested network

ECN has no effect on performance

Experimental Results — 98% Load

DCN performance

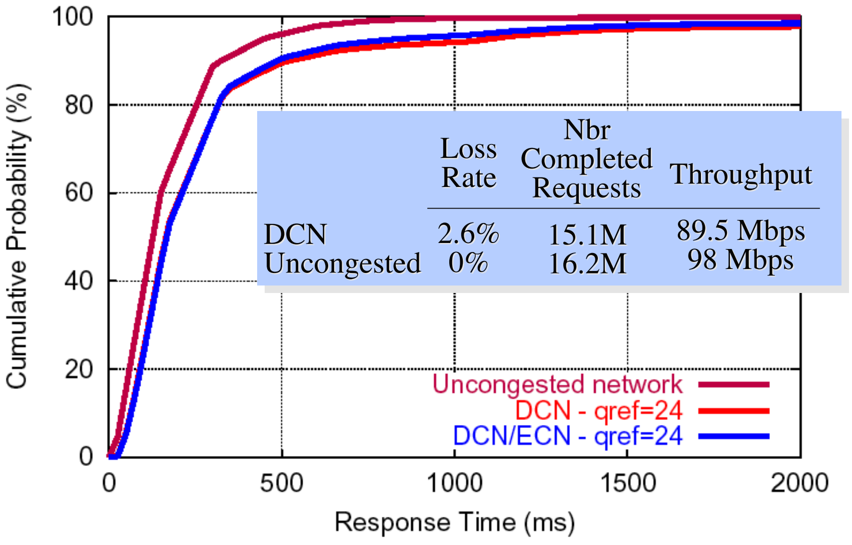


Minor (but expected) improvement with ECN

No congestion collapse effects observed

Experimental Results — 98% Load

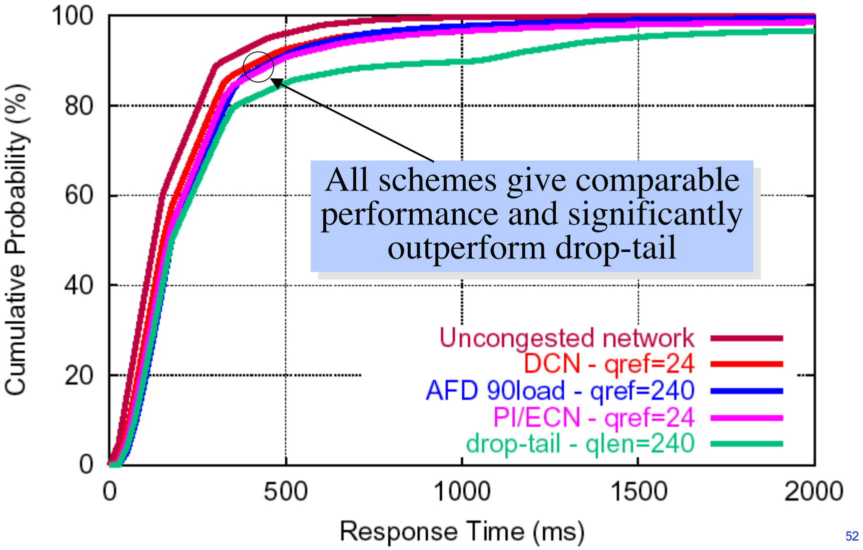
DCN performance



	Loss Rate	Nbr Completed Requests	Throughput
DCN	2.6%	15.1M	89.5 Mbps
Uncongested	0%	16.2M	98 Mbps

Experimental Results — 90% Load

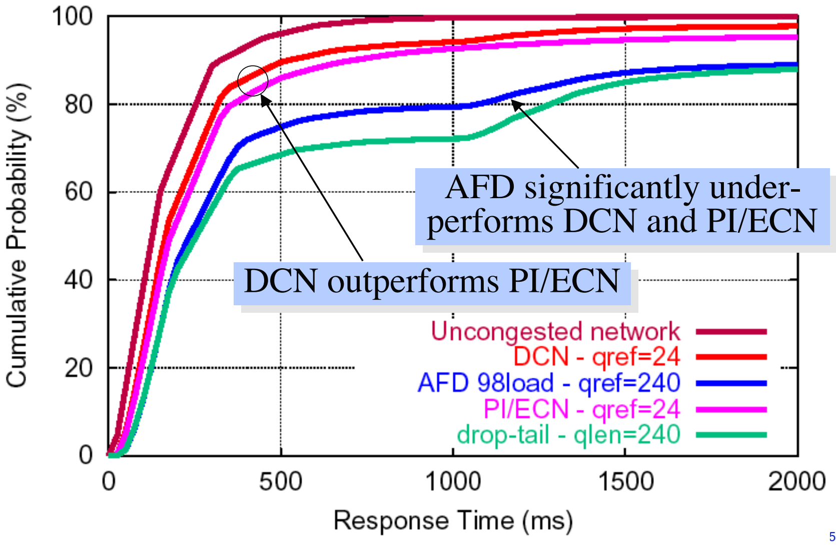
Comparison of all schemes



All schemes give comparable performance and significantly outperform drop-tail

Experimental Results — 98% Load

Comparison of all schemes

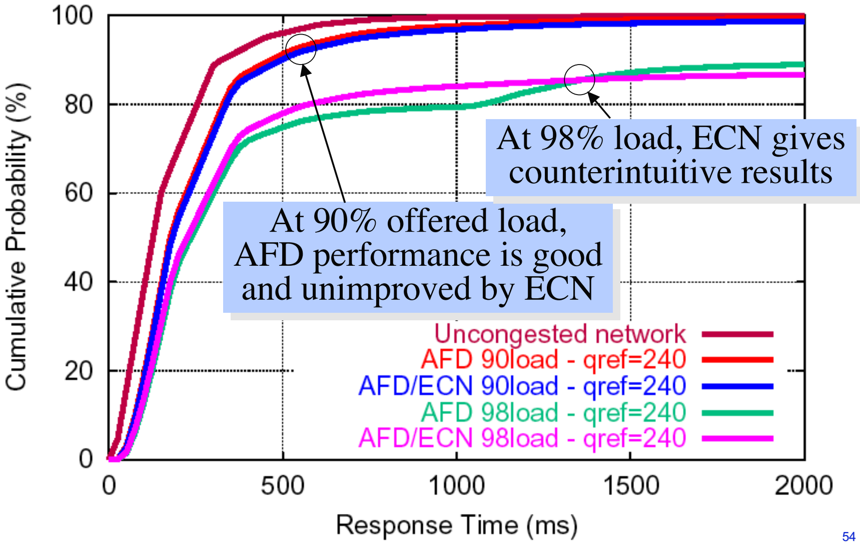


AFD significantly under-performs DCN and PI/ECN

DCN outperforms PI/ECN

Experimental Results

AFD Performance with & without ECN

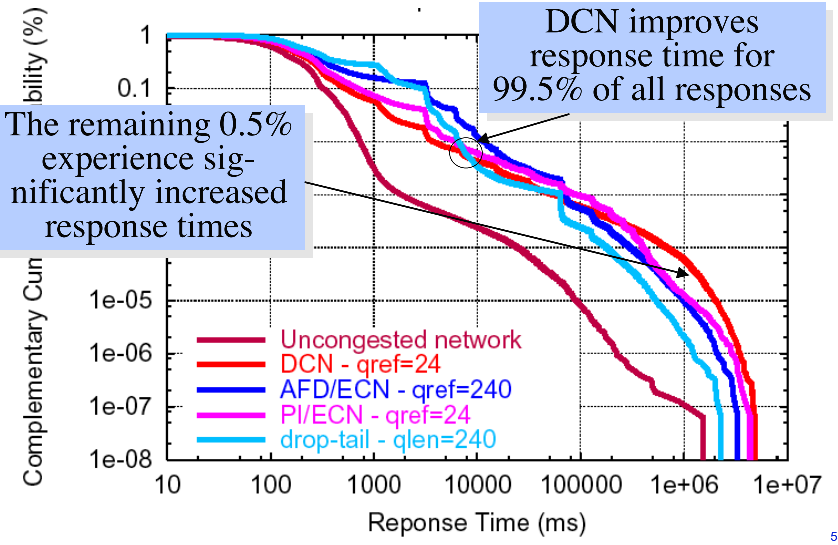


At 98% load, ECN gives counterintuitive results

At 90% offered load, AFD performance is good and unimproved by ECN

Experimental Results — 98% Load

Tail of the response time distribution



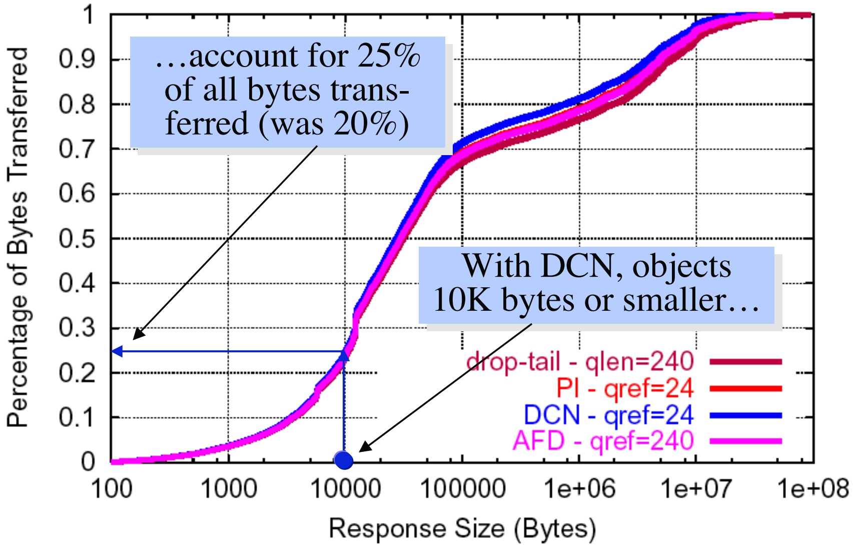
DCN improves response time for 99.5% of all responses

The remaining 0.5% experience significantly increased response times

---

Experimental Results — 98% Load

Percentage of bytes transferred by response size

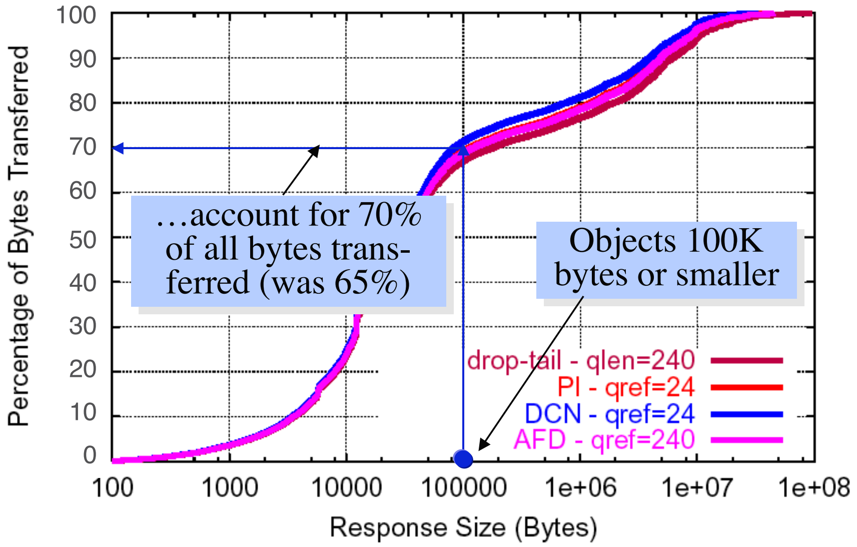


…account for 25% of all bytes transferred (was 20%)

With DCN, objects 10K bytes or smaller…

Experimental Results — 98% Load

Percentage of bytes transferred by response size



…account for 70% of all bytes transferred (was 65%)

Objects 100K bytes or smaller

DCN Evaluation

Summary

- DCN uses a simple, tunable two-tiered classification scheme with:
  - Tunable storage overhead
  - *O*(1) complexity with high probability
- DCN, without ECN, meets or exceeds the performance of the best performing AQM designs with ECN
  - The performance of 99+% of flows is improved
  - More small and "medium" flows complete per unit time
- On heavily congested networks, DCN closely approximates the performance achieved on an uncongested network

---

Making AQM Work

Summary and Conclusions

- We emulated a peering point between two ISPs and applied AQM in ISP border routers
- We emulated the browsing behaviors of tens of thousands of users in a laboratory testbed
- No AQM scheme with or without ECN is better than drop-tail FIFO for offered loads up to 80% of link capacity
- For offered loads of 90% or greater there is benefit to control theoretic AQM but only when used with ECN

Making AQM Work

Summary and Conclusions

- The reliance on ECN is required to "improve" (hurt less) the performance of short flows
  - 90% of the flows in our HTTP model
- But in the absolute, ECN is not helping their performance
- Heuristically signaling only long-lived, high-bandwidth flows improves the performance of most flows and eliminates the requirement for ECN
  - One can operate links carrying HTTP traffic at near saturation levels with performance approaching an achieved on an uncongested network
- Identification of short flows can effectively be performed with tunable state and complexity

Making AQM Work

Future work

- More of the same…
  - Tuning, tuning, tuning…
  - Re-evaluate DCN (and other AQM schemes) with more diverse traffic models

    (But where do we get these models?)
  - Study the effect of non-responsive and malicious flows
- New and improved…
  - Deconstruct AQM and study performance contribution of constituent components
  - Understand the interplay between ECN and AQM components

Making AQM Work: An Efficient Alternative to ECN

Long Le, Jay Aikat, Kevin Jeffay, and Don Smith

October 2003

http://www.cs.unc.edu/Research/dirt