Differential Congestion Notification: Taming the Elephants

Long Le Jay Aikat Kevin Jeffay F. Donelson Smith
Department of Computer Science
University of North Carolina at Chapel Hill
http://www.cs.unc.edu/Research/dirt

Abstract — Active queue management (AQM) in routers has been proposed as a solution to some of the scalability issues associated with TCP's pure end-to-end approach to congestion control. A recent study of AQM demonstrated its effectiveness in reducing the response times of web request/response exchanges as well as increasing link throughput and reducing loss rates [10]. However, use of the ECN (explicit congestion notification) signaling protocol was required to outperform drop-tail queuing. Since ECN is not currently widely deployed on end-systems, we investigate an alternative to ECN, namely applying AQM differentially to flows based on a heuristic classification of the flow's transmission rate. Our approach, called differential congestion notification (DCN), distinguishes between "small" flows and "large" high-bandwidth flows and only provides congestion notification to large high-bandwidth flows. We compare DCN to other prominent AQM schemes and demonstrate that for web and general TCP traffic, DCN outperforms all the other AQM designs, including those previously designed to differentiate between flows based on their size and rate.

1. Introduction

Congestion control on the Internet has historically been performed end-to-end with end-systems assuming the responsibility for detecting congestion and reacting to it appropriately. Currently, TCP implementations detect instances of packet loss, interpret these events as indicators of congestion, and reduce the rate at which they are transmitting data by reducing the connection's window size. This congestion reaction (combined with a linear probing congestion avoidance mechanism) successfully eliminated the occurrence of congestion collapse events on the Internet and has enabled the growth of the Internet to its current size.

However, despite this success, concerns have been raised about the future of pure end-to-end approaches to congestion control [1, 5]. In response to these concerns, router-based congestion control schemes known as active queue management (AQM) have been developed and proposed for deployment on the Internet [1]. With AQM, it is now possible for end-systems to receive a signal of incipient congestion prior to the actual occurrence of congestion. The signal can be implicit, realized by a router dropping a packet from a connection even though resources exist to enqueue and forward the packet, or the signal can be explicit, realized by a router setting an explicit congestion notification (ECN) bit in the packet's header and forwarding the packet.

In a previous study of the effects of prominent AQM designs on web performance, we argued that ECN was required in order to realize the promise of AQM [10]. This was a positive result that showed a tangible benefit to both users and service providers to deploying AQM with ECN. When compared to drop-tail routers, the deployment of particular AQM schemes with ECN (and with ECN support in end-system protocol stacks) allowed users to experience significantly reduced response times for web request/response exchanges, and allowed service providers to realize higher link utilization and lower loss rates. Without ECN, certain AQM schemes could realize modest performance improvements over simple drop-tail queue management, but the gains were small compared to those achievable with ECN.

The positive ECN results, however, beg the question of whether or not all AQM inherently requires ECN in order to be effective, or if it simply is the case that only existing AQM designs require ECN in order to be effective. This is a significant issue because ECN deployment requires the participation of both routers and end-systems and hence raises a number of issues including the cost and complexity of implementing and deploying ECN, the incremental deployability of ECN, and the (largely unstudied) issue of dealing with malicious end-systems that advertise ECN support but in fact ignore ECN signals or simply have not been configured appropriately. Other deployment issues include the fact that many firewalls and network address translators intentionally or unintentionally drop all ECN packets or clear ECN bits. In a study of TCP behavior, Padhye and Floyd found that less than 10% of the 24,030 web servers tested had ECN enabled, of which less than 1% had a compliant implementation of ECN [14]. More recent results (August 2003) showed that only 1.1% of 441 web servers tested had correctly deployed ECN [15]. This clearly points to obvious difficulties in deploying and properly using ECN on the end-systems. Thus, AQM could be significantly more appealing if ECN were not required for effective operation.

In this paper, we present an AQM design that signals congestion based on the size and rate of the flow and does not require ECN for good performance. Our approach is to differentially signal congestion to flows (through the dropping of packets) based upon a heuristic classification of the length and rate of the flow. We classify traffic into "mice," short connections that dominate on many Internet links (more than 84% of all flows in some cases [21]), and "elephants," long connections that, while relatively rare, account for the majority of bytes transferred on most links (more than 80% of all bytes [21]). Our AQM design attempts to

In: Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP),
Berlin, Germany, October 2004, pages 118-128.

notify only high-bandwidth "elephants" of congestion while allowing "slower" (and typically shorter) connections to remain unaware of incipient congestion. The motivation for this approach, borne out by our analysis of AQM schemes, is that providing early congestion notifications to mice only hurts their performance by forcing these short TCP connections to simply wait longer to transmit their last few segments. These short flows are often too short to have a meaningful transmission rate and are so short that slowing them down does not significantly reduce congestion. In contrast, providing early congestion notification to elephants can lead to abatement of congestion and more efficient use of the network. Our form of differential congestion notification, called DCN, significantly improves the performance of the vast majority of TCP transfers and provides response times, link utilizations, and loss ratios that are better than those of existing AQM schemes including those that also attempt to differentiate between flows based on their size and rate.

The remainder of the paper makes the case for differential congestion notification based on classification of flow-rate. Section 2 discusses previous related work in AQM schemes in general and in differential AQM specifically. Section 3 presents our DCN scheme. Section 4 explains our experimental evaluation methodology and Section 5 presents the results of a performance study of DCN and several prominent AQM schemes from the literature. The results are discussed in Section 6.

2. Background and Related Work

Several AQM designs have attempted to achieve fairness among flows or to control high-bandwidth flows. Here we give a description of the AQM designs most related to ours.

The Flow Random Early Drop (FRED) algorithm protects adaptive flows from unresponsive and greedy flows by implementing per-flow queueing limits [11]. The algorithm maintains state for each flow that currently has packets queued in the router. Each flow is allowed to have up to min_q but never more than max_q packets in the queue. Packets of flows that have more than min_q but less than max_q packets in the queue are probabilistically dropped. When the number of flows is large and a high-bandwidth flow consumes only a small fraction of the link capacity, FRED maintains a large queue. However, a large queue results in high delay. Furthermore, the algorithm becomes less efficient with a large queue since the search time for flow state is proportional to queue length.

The Stabilized Random Early Drop (SRED) algorithm controls the queue length around a queue threshold independent of the number of active connections [13]. The algorithm keeps the header of recent packet arrivals in a "zombie list." When a packet arrives, it is compared with a randomly chosen packet from the zombie list. If the two packets are of the same flow, a "hit" is declared. Otherwise, the packet header in the zombie list is probabilistically replaced by the header of the new packet. The number of active connections is estimated as the reciprocal of the average number of hits in a given interval (a large number of active connections results in a low probability of hits and vice versa). The drop probability of a packet is a function of the instantaneous queue length and the estimated number of active connections. Hits are also used to identify high-bandwidth flows. Packets of high-bandwidth flows are dropped with a higher probability than other packets.

The CHOKe algorithm heuristically detects and discriminates against high-bandwidth flows without maintaining per flow state [17]. The algorithm is based on the assumption that a high-bandwidth flow is likely to occupy a large amount of buffer space in the router. When a new packet arrives, CHOKe picks a random packet in the queue and compares that packet's header with the new packet's header. If both packets belong to the same flow, both are dropped, otherwise, the new packet is enqueued. As with FRED, CHOKe is not likely to work well on a high-speed link and in the presence of a large aggregate of flows.

The Stochastic Fair BLUE (SFB) algorithm detects and rate-limits unresponsive flows by using accounting bins that are organized hierarchically [4]. The bins are indexed by hash keys computed from a packet's IP addresses and port numbers and used to keep track of queue occupancy statistics of packets belonging to the bin. High-bandwidth flows can be easily identified because their bins' occupancy is always high. These high-bandwidth flows are then rate-limited. SFB works well when the number of high-bandwidth flows is small. When the number of high-bandwidth flows increases, more bins become occupied and low bandwidth flows that hash to these bins are incorrectly identified as high-bandwidth and penalized.

The Approximate Fairness through Differential Dropping (AFD) algorithm approximates fair bandwidth allocation by using a history of recent packet arrivals to estimate a flow's transmission rate [16]. AFD uses a control theoretic algorithm borrowed from PI [7] to estimate the "fair share" of bandwidth that a flow is allowed to send. Packets of a flow are marked or dropped with a probability that is a function of the flow's estimated sending rate. The algorithm uses a *shadow buffer* to store recent packet headers and uses these to estimate a flow's rate. The estimated rate of a flow is proportional to the number of that flow's headers in the shadow buffer. When a packet arrives, its header is copied to the shadow buffer with probability $1/s$, where s is the sampling interval, and another header is removed randomly from the shadow buffer. Note that while sampling reduces implementation overhead, it also reduces the accuracy in estimating flows' sending rate. This problem can be severe when most flows only send a few packets per RTT.

The RED with Preferential Dropping (RED-PD) algorithm provides protection for responsive flows by keeping state

for just the high-bandwidth flows and preferentially dropping packets of these flows [12]. RED-PD uses the history of recent packet drops to identify and monitor high-bandwidth flows. The algorithm is based on the assumption that high-bandwidth flows also have a high number of packet drops in the drop history. Packets of a high-bandwidth flow are dropped with a higher probability than other packets. After being identified as high-bandwidth, a flow is monitored until it does not experience any packet drop in a certain time period. The absence of packet drops of a high-bandwidth flow in the drop history indicates that the flow has likely reduced its sending rate. In this case, the flow is deleted from the list of monitored flows.

The RIO-PS scheme (Red with In and Out with Preferential treatment to Short flows) gives preferential treatment to short flows at bottleneck links [6]. With preferential treatment, short flows experience a lower drop-rate than long flows and can thus avoid timeouts. In RIO-PS, edge routers maintain per-flow state for flows entering the network. The first few packets of a flow are marked as "short" or "in." Subsequent packets of that flow are marked as "long" or "out." Core routers use the standard RIO algorithm [2] and drop long or out packets with a higher probability than short or in packets.

Our DCN algorithm, described next, is an amalgam of existing AQM mechanisms. Like AFD it uses a control theoretic algorithm for selecting packets to drop and like RED-PD it maintains state for only the suspected high-bandwidth flows. However, we show empirically that our particular choice and construction of mechanisms results in better application and network performance than is possible with existing differential and non-differential AQM designs.

3. The DCN Algorithm

The design of DCN is based on the observation that on many networks, a small number of flows produce a large percentage of the traffic. For example, for the web traffic we have used to evaluate AQM designs, Figure 1 shows the cumulative distribution function (CDF) of the empirical distribution of HTTP response sizes [18]. Figure 2 shows a CDF of the percentage of total bytes transferred in an hour-long experiment as a function of HTTP response size. Together, these figures show that while approximately 90% of web responses are 10,000 bytes or less, these responses account for less than 25% of the bytes transferred during an experiment. Moreover, responses greater than 1 megabyte make up less than 1% of all responses, but account for 25% of the total bytes.

These data suggest that providing early congestion notification to flows carrying responses consisting of a few TCP segments (*e.g.*, flows of 2,000-3,000 bytes, approximately 70% of all flows), would have little effect on congestion. This is because these flows comprise only 6-8% of the total bytes and because these flows are too short to have a transmission rate that is adaptable. By the time they receive a congestion notification signal they have either already completed or have only one segment remaining to be sent. Furthermore, since short flows have a small congestion window, they have to resort to timeouts when experiencing a packet loss. Thus giving these flows a congestion signal does not significantly reduce congestion and can only hurt the flows' performance by delaying their completion. In contrast, high-bandwidth flows carrying large responses are capable of reducing their transmission rate and hence can have an impact on congestion. Unlike short flows, high-bandwidth flows do not have to resort to timeouts and instead can use TCP mechanisms for fast retransmission and fast recovery to recover from their packet losses. Our approach will also police high-bandwidth non-TCP or non-compliant TCP flows that do not reduce their transmission rate when congestion occurs.

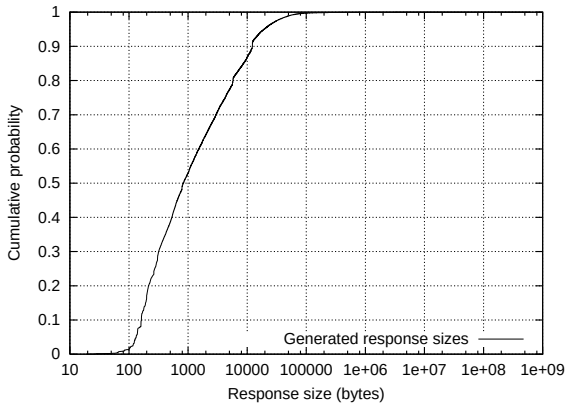

Figure 1: CDF of generated response sizes.

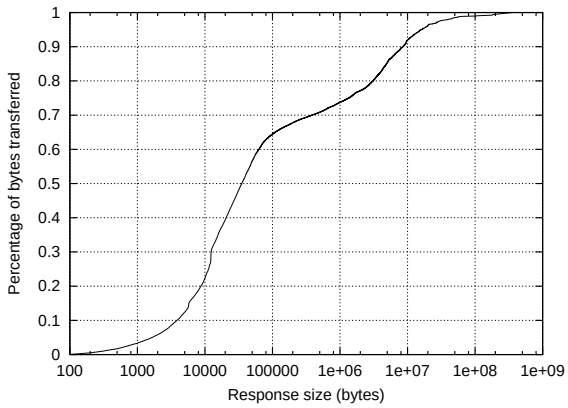

Figure 2: CDF of percentage of total bytes transferred as a function of response sizes.

Our observation about traffic characteristics is also confirmed by other studies of Internet traffic. For example, Zhang *et al.* found that small flows (100KB or less) accounted for at least 84% of all flows, but carried less than 15% of all bytes [21]. They also found that large flows accounted for a small fraction of the number of flows, but carried most of the bytes. Moreover, the flows that are "large"

and "fast" (*i.e.*, high-bandwidth, greater than 10KB/sec) account for less than 10% of all flows, but carrying more than 80% of all the bytes.

The Differential Congestion Notification (DCN) scheme is based on identifying these "large" and "fast" flows, and providing congestion notification to only them. While the idea is simple, the challenge is to design an algorithm with minimal state requirements to identify the few long-lived, high-bandwidth flows from a large aggregate of flows and provide them with a congestion signal when appropriate. An important dimension of this problem is that of all the flows carrying large responses, we most want to signal flows that are also transmitting at a high-rate. These are the flows that are consuming the most bandwidth and hence will produce the greatest effect when they reduce their rate. Additionally, we must ensure that flows receiving negative differential treatment are not subject to undue starvation.

Our DCN AQM design has two main components: identification of high-bandwidth flows and a decision procedure for determining when early congestion notification is in order.

3.1 Identifying High-bandwidth Flows

Our approach to identifying high-bandwidth, long-lived flows is based on the idea that packets of high-bandwidth flows are closely paced (*i.e.*, their interarrival times are short) [3]. DCN tracks the number of packets that have been recently seen from each flow. If this count exceeds a threshold, the flow is considered to be a "long-lived and high-bandwidth" flow. The flow's rate is then monitored and its packets are eligible for dropping. As long as a flow remains classified as high-bandwidth, it remains eligible for dropping. If a flow reduces its transmission rate, it is removed from the list of monitored flows and is no longer eligible for dropping.

DCN uses two hash tables for classifying flows: HB ("high bandwidth") and SB ("scoreboard"). The HB table tracks flows that are considered high-bandwidth and stores each flow's flow ID (IP addressing 5-tuple) and the count of the number of forwarded packets. The SB table tracks a fixed number of flows not stored in HB. For these flows SB stores their flow ID and "recent" forwarded packet count.

When a packet arrives at a DCN router, the HB table is checked to see if this packet belongs to a high-bandwidth flow. If the packet's flow is found in HB, then it is handled as described below. If the packet's flow ID is not in HB, then the packet is enqueued and its flow is tested to see if it should be entered into HB. The SB table is searched for the flow's ID. If the flow ID is not present, it is added to SB.[1] If the flow ID is present in SB, the flow's packet count is incremented.

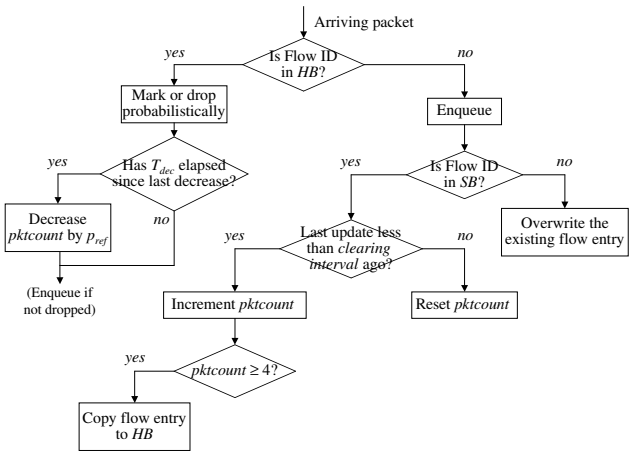

Figure 3: High-level DCN flowchart.

A flow is classified as long-lived and high-bandwidth if the number of packets from the flow arriving within a "clearing interval," exceeds a threshold. Once the flow's packet count in SB has been incremented, if the count exceeds the threshold, the flow's entry in SB is added to HB.[2] If no packets have been received for the flow within a clearing interval, the flow's packet count is reset to 0.

A high-level flow chart of the DCN algorithm is given in Figure 3. All operations on the SB table are performed in $O(1)$ time. Since the number of flows identified as high-bandwidth is small (*e.g.*, ~2000 for traffic generated during experiments reported herein), hash collisions in HB are rare for a table size of a few thousand entries. Thus, operations on the HB table are also usually executed in $O(1)$ time.

3.2 Early Congestion Notification

Packets from a high-bandwidth flow are dropped with a probability $1 - p_{ref} / pktcount$, where *pktcount* is the number of packets from that flow that have arrived at the router within a period of T_{dec}, and p_{ref} is the current "fair share" of a flow on a congested link. When congestion is suspected in the router we target high-bandwidth flows for dropping in proportion to their deviation from their fair share (p_{ref} packets within an interval T_{dec}) [16, 19].

DCN uses a simple control theoretic algorithm based on the well-known proportional integral controller to compute p_{ref}. The instantaneous length of the queue in the router is periodically sampled with period T_{update}. A flow's fair share of the queue at the k^{th} sampling period is given by:

$$p_{ref}(kT_{update}) = p_{ref}((k-1)T_{update}) + a \times (q(kT_{update}) - q_{ref}) - b \times (q((k-1)T_{update}) - q_{ref})$$

[1] If a flow ID hashes to an entry in SB for another flow, then the new flow overwrites the entry for the previous flow. This ensures that all SB operations can be performed in constant time. Thus, SB stores the packet counts for all currently active non-high-bandwidth flows, modulo hash function collisions on flow IDs.

[2] If a collision occurs in HB when trying to insert the new flow, then a hash chain is used to locate a free table entry.

where a and b, $a < b$, are control coefficients (constants) that depend on the average number of flows and the average RTT of flows (see [7] for a discussion), $q()$ is the length of the queue at a given time, and q_{ref} is a target queue length value for the controller. Since $a < b$, p_{ref} decreases when the queue length is larger than q_{ref} (an indication of congestion) and hence packets from high-bandwidth flows are dropped with a high probability. When congestion abates and the queue length drops below q_{ref}, p_{ref} increases and the probability of dropping becomes low. Pan *et al.* and Misra *et al.* use the same equation in the design of AFD and PI respectively [16, 7].

The flow ID of a high-bandwidth flow is kept in the HB table as long as the flow's counter *pktcount* is positive. After each interval T_{dec}, the counter *pktcount* is decreased by p_{ref}. If a high-bandwidth flow's packet count becomes negative, the flow is deleted from HB. We set T_{dec} to 800 *ms* in our experiments because the maximum RTT in our network can be up to 400 *ms*. Furthermore, we want to avoid situations where a new high-bandwidth flow is detected at the end of an interval T_{dec} and immediately removed from HB. We experimented with different parameter settings for a, b, T_{update}, and T_{dec} and here report only the results for our empirically determined best parameter settings.

4. Experimental Methodology

To evaluate DCN we ran experiments in the testbed network described in [10]. The network, illustrated in Figure 4, emulates a peering link between two Internet service provider (ISP) networks. The testbed consists of approximately 50 Intel processor based machines running FreeBSD 4.5. Machines at the edge of the network execute one of a number of synthetic traffic generation programs described below. These machines have 100 Mbps Ethernet interfaces and are attached to switched VLANs with both 100 Mbps and 1 Gbps ports on 10/100/1000 Ethernet switches. At the core of this network are two router machines running the ALTQ extensions to FreeBSD [9]. ALTQ extends IP-output queuing at the network interfaces to include alternative queue-management disciplines. We used the ALTQ infrastructure to implement DCN, AFD, RIO-PS, and PI.

Each router has sufficient network interfaces to create either a point-to-point 100 Mbps or 1 Gbps Ethernet network between the two routers. The Gigabit Ethernet network is used to conduct calibration experiments to benchmark the traffic generators on an unloaded network. To evaluate DCN and compare its performance to other AQM schemes, we create a bottleneck between the routers by altering the (static) routes between the routers so that all traffic flowing in each direction uses a separate 100 Mbps Ethernet segment. This setup allows us to emulate the full-duplex behavior of a typical wide-area network link.

So that we can emulate flows that traverse a longer network path than the one in our testbed, we use a locally-modified

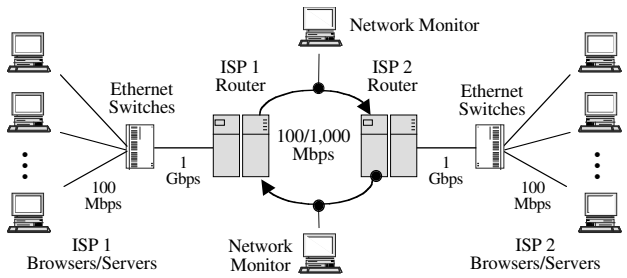

Figure 4: Experimental network setup.

version of *dummynet* [8] to configure out-bound packet delays on machines on the edge of the network. These delays emulate different round-trip times on *each* TCP connection (thus giving *per-flow* delays). Our version of *dummynet* delays all packets from each flow by the same randomly-chosen minimum delay. The minimum delay in milliseconds assigned to each flow is sampled from a discrete uniform distribution on the range [10, 150] with a mean of 80 *ms*. The minimum and maximum values for this distribution were chosen to approximate a typical range of Internet round-trip times within the continental U.S. and the uniform distribution ensures a large variance in the values selected over this range.

A TCP window size of 16K bytes was used on the end systems because widely used OS platforms, *e.g.*, most versions of Windows, typically have default windows of 16K or less.

4.1 Synthetic Generation of TCP Traffic

Two synthetically generated TCP workloads will be used to evaluate DCN. The first is an HTTP workload derived from a large-scale analysis of web traffic [18]. Synthetic HTTP traffic is generated according to an application-level description of how the HTTP 1.0 and 1.1 protocols are used by web browsers and servers today. The specific model of synthetic web browsing is as described in [10], however, here we note that the model is quite detailed as it, for example, includes the use of persistent HTTP connections and distinguishes between web objects that are "top-level" (*e.g.*, HTML files) and objects that are embedded (*e.g.*, image files).

The second workload is based on a more general model of network traffic derived from measurements of the full mix of TCP connections present on Internet links. For the experiments here we emulate the traffic observed on an Internet 2 backbone link between Cleveland and Indianapolis [20]. Thus in addition to generating synthetic HTTP connections, this model will also generate synthetic FTP, SMTP, NNTP, and peer-to-peer connections. Details on this model can be found in [20].

For both workloads, end-to-end response times for TCP data exchanges will be our primary measure of performance. Response time is defined as the time interval necessary to complete the exchange of application-level data units be-

tween two endpoints. For example, for the HTTP workload, response time is defined as the time between when an (emulated) web browser makes a request for content to an (emulated) web server and the time when the last byte of the response is delivered from the server to the browser.

4.2 Experimental Procedures

To evaluate DCN we performed experiments on the two emulated ISP networks in our testbed connected with 100 Mbps links. For the evaluation we congest these 100 Mbps links with varying degrees of traffic. To quantify the traffic load in each experiment we define *offered load* as the network traffic (in bits/second) resulting from emulating the behavior of a fixed-size population of users (*e.g.*, a population of users browsing the web in the case of the HTTP workload). More specifically, load is expressed as the long-term average throughput on an uncongested link that would be generated by that user population. For example, to describe the load offered by emulating a population of 20,000 users evenly distributed on our network testbed, we would first emulate this user population with the two ISP networks connected with a gigabit/second link and measure the average throughput in one direction on this link. The measured throughput, approximately 105 Mbps in the case of the HTTP workload, is our value of offered load.

Although the TCP traffic we generate is highly bursty (*e.g.*, for the HTTP workload we generate a long-range dependent packet arrival process at the routers [10]), the range of loads we attempt to generate in our experiments (approximately 50-120 Mbps) is such that congestion will never be present on the gigabit network. Since experiments are ultimately performed with the two ISP networks connected at 100 Mbps, a calibration process (described in [10]) is used to determine the population of users required to achieve a particular degree of congestion on the 100 Mbps network.

Each experiment was run using offered loads of 90%, 98%, or 105% of the capacity of the 100 Mbps link connecting the two router machines. It is important to emphasize that terms like "105% load" are used as a shorthand notation for "a population of users that would generate a long-term average load of 105 Mbps on a 1 Gbps link." As offered loads approach saturation of the 100 Mbps link, the actual link utilization will, in general, be less than the intended offered load. This is because as utilization increases, response times become longer and emulated users have to wait longer before they can generate new requests and hence generate fewer requests per unit time.

Each experiment was run for 120 minutes to ensure very large samples (over 10,000,000 TCP data exchanges), but data were collected only during a 90-minute interval to eliminate startup effects at the beginning and termination synchronization anomalies at the end.

The key indicator of performance we use in reporting our results are the end-to-end response times for each data exchange (*e.g.*, an HTTP request/response). We report these as CDFs of response times up to 2 seconds. We also report the fraction of IP datagrams dropped at the link queues, the link utilization on the bottleneck link, and the number of request/response exchanges completed in the experiment.

5. Experimental Results

We compared the performance of DCN against a large number of AQM designs but report here only the results of DCN versus only PI, AFD and RIO-PS. PI was selected because it was the best performing non-differential AQM scheme from a previous study [10]. AFD and RIO-PS were chosen because they were the best performing differential schemes. We also include results from experiments with drop-tail FIFO queue management to illustrate the performance of no AQM (*i.e.*, the performance to beat), and results from drop-tail experiments on the uncongested gigabit network to illustrate the best possible performance. (The experiments on the uncongested network provide the best possible response times as there is no queue present on either router at any of the load levels considered here.)

For PI, DCN and AFD, an important parameter is the target queue length the algorithm attempts to maintain. After extensive experimentation, we found that AFD perform best at a target queue length of 240 packets at all loads. DCN and PI obtain their best performance with a target queue length of 24 packets. We used the parameter settings for RIO-PS that were recommended by its inventors [6]. In all cases we set the maximum queue size to a number of packets that was large enough to ensure tail drops did not occur.

5.1 Experimental Results for the HTTP Workload

Figures 5-8 give the results for DCN, PI, AFD, and RIO-PS. They show the cumulative distribution functions (CDFs) for response times of HTTP request/response exchanges at offered loads of 90%, 98%, and 105% respectively. We also report other statistics for the experiments in Table 1.

Figure 5 shows that at all loads, the performance of HTTP flows under DCN is close to that obtained on the uncongested network. Although DCN performance decreases as load increases, the performance degradation of DCN is rather small and DCN still obtains good performance for flows even at very high load (98% and 105%).

Figure 6 gives the comparable results for PI (reproduced here from [10] but obtained under identical experimental conditions). Without ECN, the response-time performance obtained with PI degrades rapidly with loads above 90%. However, with ECN, even at loads of 98% and 105% performance is only modestly worse than it is at 90% load and is reasonably comparable to the uncongested case.

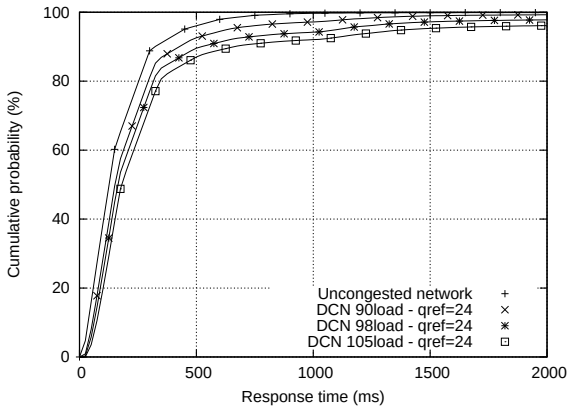

Figure 5: DCN response time CDF for HTTP request/response exchanges for various offered loads.

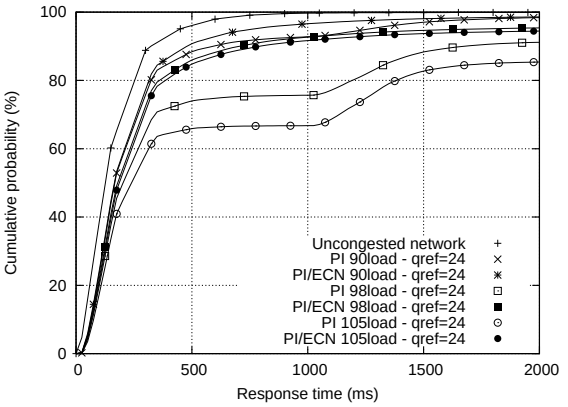

Figure 6: PI response time CDF for HTTP request/response exchanges for various offered loads.

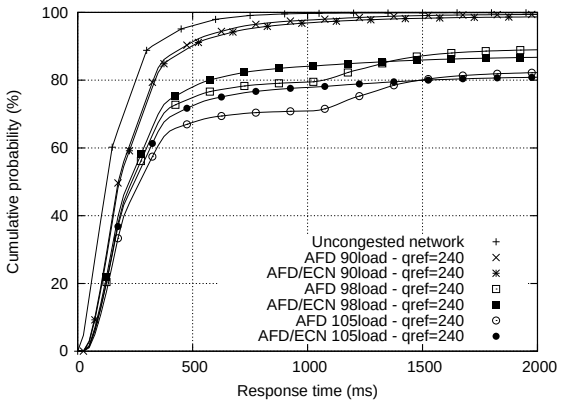

Figure 7: AFD response time CDF for HTTP request/response exchanges for various offered loads.

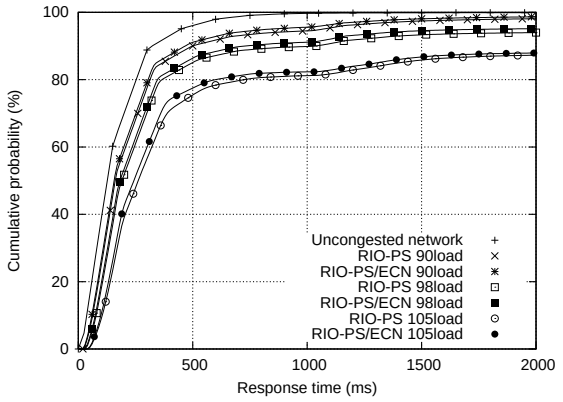

Figure 8: RIO-PS response time CDF for HTTP request/response exchanges for various offered loads.

Figure 7 shows the results for AFD. At 90% load, the performance of AFD is fairly close to that of an uncongested network and there is almost no performance gain with the addition of ECN. There is significant degradation in response-time performance as load is increased to 98% and 105%. AFD does show a small yet noticeable improvement with addition of ECN at these loads. This is because with ECN, some connections avoid a timeout that otherwise would have resulted from drops by the AFD algorithm. This can be seen by examining the CDFs for the non-ECN cases and observing show a flat region extending out to 1 second (the minimum timeout in FreeBSD's TCP implementation).

Figure 8 shows the results for RIO-PS. RIO-PS obtains reasonably good performance at 90% load and experiences more modest (but still significant) performance degradation at higher loads. At all loads, ECN contributes little to the performance of flows under RIO-PS.

A more direct comparison of these AQM designs is shown in Figures 9, 11, and 13, which show response time performance of the best performing version of each AQM design at loads of 90%, 98%, and 105%, respectively. We also include in these plots the results of experiments with drop-tail on the same congested 100 Mbps network.

At 90% load all the algorithms except drop-tail perform quite well and provide response times that are not significantly degraded over the uncongested case. At loads of 98% and 105% there is clear performance superiority for DCN and PI with ECN over the other AQMs. However, in both cases, DCN, even without ECN, provides performance that is equal or superior to PI with ECN.

For the vast majority of flows, the best AQM designs provide performance superior to drop-tail at all loads. This result demonstrates the benefits of AQM. However, the results also show that differential treatment of flows is not a panacea. PI with ECN's uniform treatment of flows provides comparable or better performance for flows than AFD's and RIO-PS's differential treatment.

Comparing just the differential AQM designs, DCN obtains better performance than RIO-PS which in turn provides better performance than AFD. This reflects different cost/performance tradeoffs in differential AQM designs. AFD maintains a fixed size hash table and performs flow classification in $O(1)$ time. However, AFD has to resort to

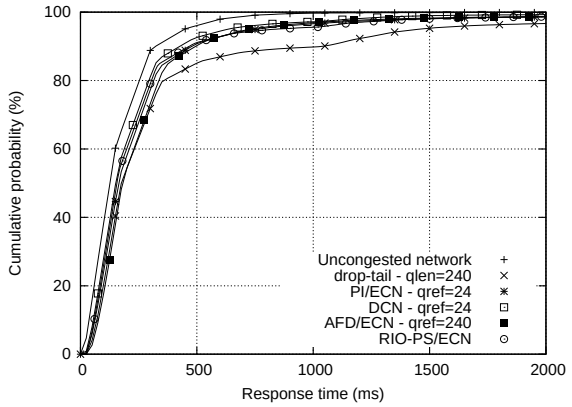

Figure 9: Comparison of all schemes at 90% load.

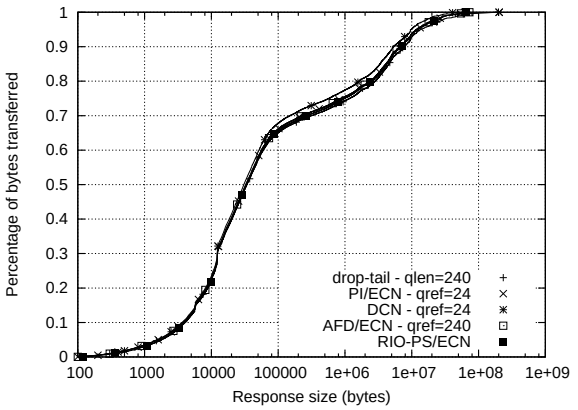

Figure 10: Distribution of the fraction of total bytes transferred as a function of response size, 90% load.

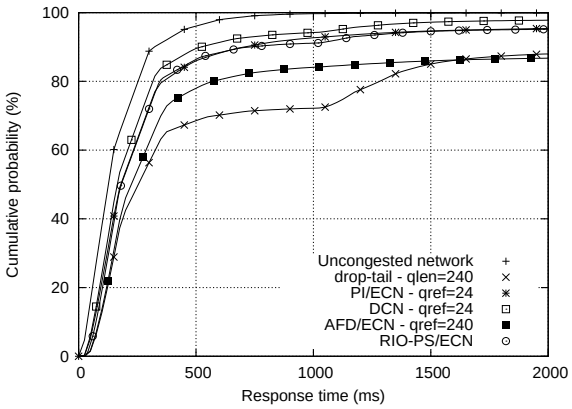

Figure 11: Comparison of all schemes at 98% load.

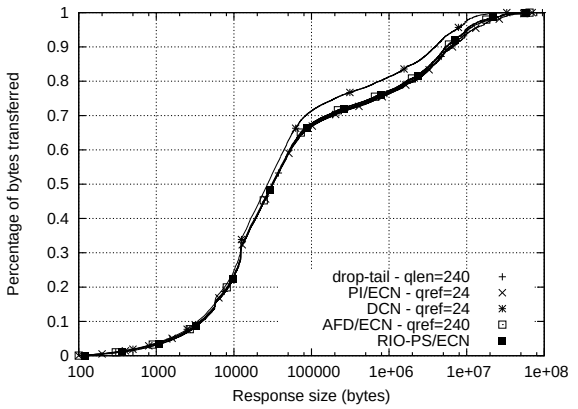

Figure 12: Distribution of the fraction of total bytes transferred as a function of response size, 98% load.

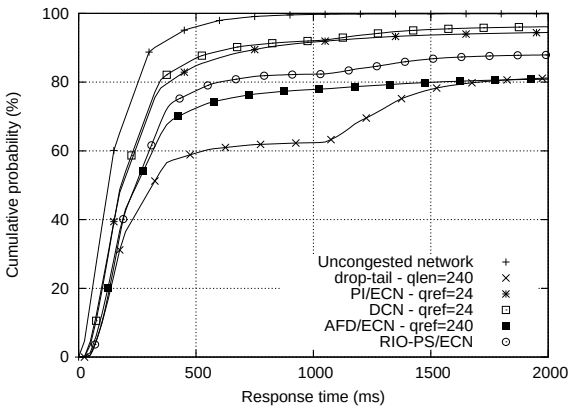

Figure 13: Comparison of all schemes at 105% load.

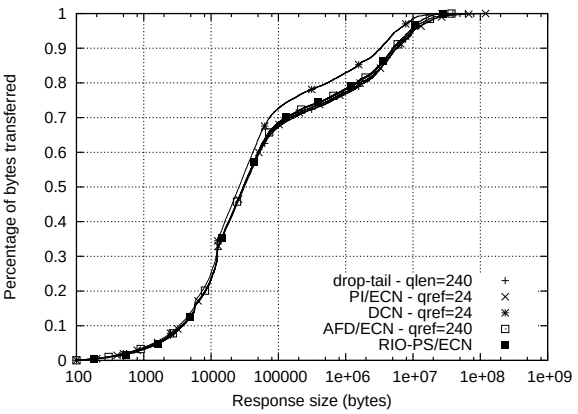

Figure 14: Distribution of the fraction of total bytes transferred as a function of response size, 105% load.

random sampling and can only detect a subset of the high-bandwidth flows. RIO-PS maintains per-flow state for all flows and has the potential for detecting all high-bandwidth flows. However, since RIO-PS classifies flows based on their size, persistent HTTP connections may be erroneously classified as high-bandwidth flows. In this case, short request/response exchanges over these persistent connections suffer high packet loss rates and bad performance. DCN, on other hand, classifies flows based on the interarrival times of packets within a flow and has a lower rate of false positives than RIO-PS. Moreover, DCN only maintains per-flow state for high-bandwidth flows and can potentially require significantly less state than RIO-PS.

We also see in Table 1 that as the load increases, the loss rate increases significantly for AFD and RIO-PS, even with

the use of ECN. With DCN's scheme for detecting and controlling high-bandwidth flows, a fairly low loss rate is maintained even at very high loads.

Simply counting the number of responses that receive better response times (as in the CDF plots) does not provide a complete picture because it does not consider which responses constitute the majority of bytes carried by the network. Figures 10, 12, and 14 give the fractions of the total bytes in all HTTP responses that are contained in responses of a specific size or smaller with the different AQM designs.

For example, we see in Figure 10 that at 90% load, approximately 20% of the total bytes are contained in responses of 10,000 bytes or less and approximately 75% of the total bytes are contained in responses of 1 MB or less. Recall, however, that in Figure 1 we saw that approximately 90% of all HTTP responses are 10,000 bytes or less and more than 99% of all responses are 1 MB or less. Because DCN controls high-bandwidth flows and protects small flows, we would expect that DCN would cause a greater fraction of the total bytes in all responses to come from the shorter responses that DCN favors (or conversely, a smaller fraction from the longer responses that are subject to DCN drops). This effect is seen in Figures 10, 12, and 14 where the curve for DCN is always above the others for response sizes greater than approximately 50,000 bytes. Thus under DCN, more shorter flows complete and they complete quicker than under the other AQM schemes.

To demonstrate the differential treatment for "small" and "large" flows by differential AQM designs, we show the CDFs of response times for objects smaller than 100,000 bytes in Figures 15 and 17, and CDFs of response times (on a log scale) for objects larger than 100,000 bytes in Figures 16 and 18. In Figures 15 and 17, we see that DCN clearly favors "small" flows as expected and outperforms the other AQM schemes at 90% and 98% load. In Figure 16, we see that DCN outperforms other AQM schemes for approximately 75% of flows that are larger than 100,000 bytes at 90% load. Similarly, we observe in Figure 18 that about 40% of flows larger than 100,000 bytes experience better performance under DCN than under other AQM schemes. Furthermore, we note that the large flows that are affected adversely by DCN already experience long response times under other AQM schemes (at least 10 seconds). Recall from Figure 1 that flows with object sizes greater than 100,000 bytes comprise less than 1% of the total number of flows. Hence, more than 99% of the flows experience performance improvement under DCN at the cost of a degradation in performance for a small number of flows.

We note that the differential treatment of DCN and RIO-PS has the same spirit as the "shortest job first" (SJF) scheduling algorithm which is provably (with regard to an abstract formal model) optimal for minimizing mean response times. While this intuition is straightforward, the challenge is to design a light-weight algorithm that can classify and control high-bandwidth flows with a low implementation overhead.

Table 1: Loss, completed requests, and link utilizations for experiments with the HTTP workload.

	Offered Load	Loss ratio (%)		Completed requests (millions)		Link utilization/ throughput (Mbps)	
		No ECN	ECN	No ECN	ECN	No ECN	ECN
Uncongested 1 Gbps network (drop-tail)	90%	0		15.0		91.3	
	98%	0		16.2		98.2	
	105%	0		17.3		105.9	
drop-tail queue size = 240	90%	1.8		14.6		89.9	
	98%	6.0		15.1		92.0	
	105%	8.8		15.0		92.4	
PI $q_{ref} = 24$	90%	1.3	0.3	14.4	14.6	87.9	88.6
	98%	3.9	1.8	15.1	14.9	89.3	89.4
	105%	6.5	2.5	15.1	15.0	89.9	89.5
DCN $q_{ref} = 24$	90%	0.95		14.5		86.1	
	98%	2.3		14.9		88.6	
	105%	3.0		15.3		90.5	
AFD $q_{ref} = 240$	90%	0.5	0.5	14.7	14.6	88.3	88.3
	98%	6.1	6.2	14.5	14.5	87.6	87.9
	105%	8.9	9.2	14.6	14.4	87.8	87.9
RIO-PS	90%	2.1	1.3	14.6	14.6	88.8	87.9
	98%	5.3	4.7	15.0	15.2	91.2	91.0
	105%	8.2	7.9	15.3	15.3	91.9	91.9

5.2 Results for the General TCP Workload

While results for DCN from previous experiments are encouraging, they are limited to only Web traffic. To demonstrate the generality of our approach, we perform a set of experiments with DCN and other AQM schemes using synthetic traffic that is derived from the full mix of TCP connections captured on Internet links.

Figures 19 and 20 show the cumulative distribution functions (CDFs) and complementary CDF (CCDFs) of response times for these experiments. Here the offered load is asymmetric with a load on the "forward" path of 105.3 Mbps and 91.2 Mbps on the "reverse" path.

We again see the benefits of differential AQM approaches, in particular with DCN and RIO-PS. These two AQM designs outperform drop-tail and other AQM designs and come close to the performance of the uncongested network. The summary statistics for these experiments are included in Table 2.

6. Summary and Conclusions

Our recent study of AQM schemes [10] demonstrated that AQM can be effective in reducing the response times of web request/response exchanges, increasing link throughput,

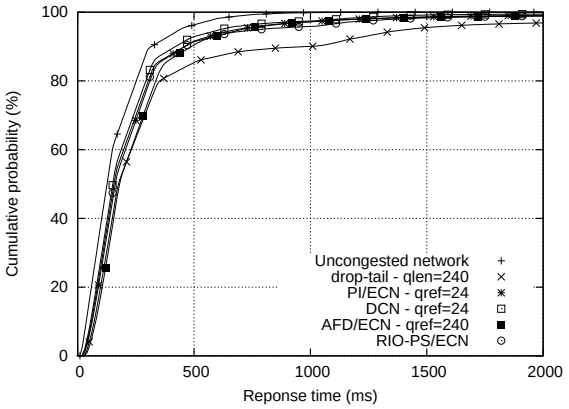

Figure 15: Distribution of response times for HTTP responses smaller than 100,000 bytes at 90% load.

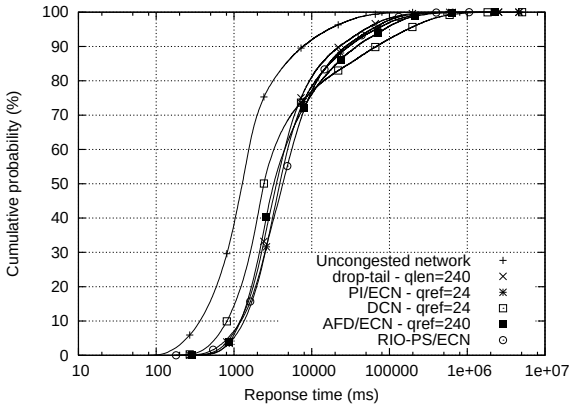

Figure 16: Distribution of response times for HTTP responses larger than 100,000 bytes at 90% load.

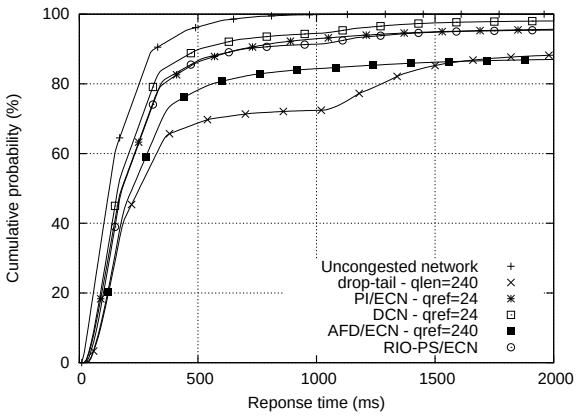

Figure 17: Distribution of response times for HTTP responses smaller than 100,000 bytes at 98% load.

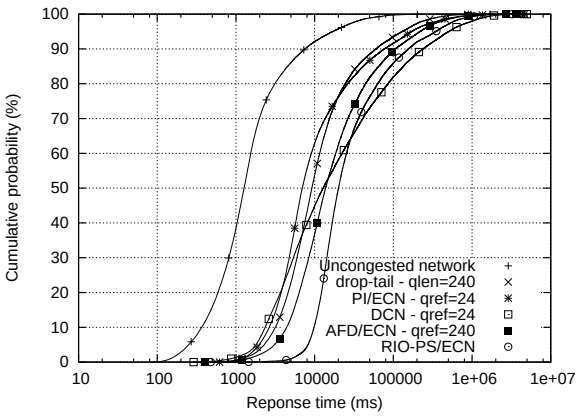

Figure 18: Distribution of response times for HTTP responses larger than 100,000 bytes at 98% load.

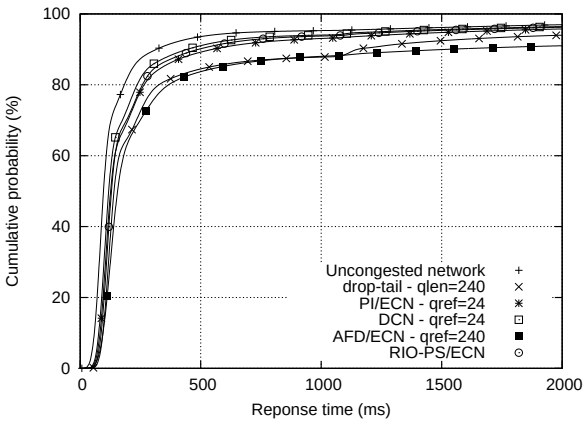

Figure 19: Distribution of response times for the general TCP workload.

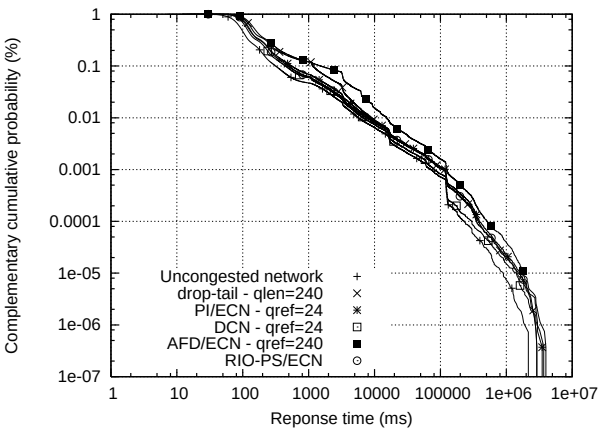

Figure 20: CCDFs of response times for the general TCP workload.

and reducing loss rates. However, ECN was required for these results. In this work we have investigated the use of differential congestion notification as a means of eliminating the requirement for ECN. Our differential AQM design, DCN, uses a two-tiered flow filtering approach to heuristically classifying flows as being long-lived and high-bandwidth or not, and provides early congestion notification to only these suspect flows. DCN can perform classification in $O(1)$ time with high probability and with a constant (and tunable) storage overhead.

We compared DCN to PI/ECN, AFD/ECN, and RIO-PS/ECN and demonstrated that DCN, without ECN, outperforms the other AQM designs. Under a variety of workloads and congestion conditions, response times for TCP data

Table 2: Summary statistics for experiments with general TCP workload

	Completed exchanges (millions)	Forward path loss rate (%)	Reverse path loss rate (%)	Forward path link throughput (Mbps)	Reverse path link throughput (Mbps)
Uncongested	2.75	0.0	0.0	105.3	91.2
drop-tail qlen = 240	2.62	3.7	0.9	90.8	85.7
PI/ECN q_{ref} = 240	2.69	0.2	0.1	88.5	84.4
DCN q_{ref} = 24	2.60	1.0	0.6	89.4	83.3
AFD/ECN q_{ref} = 240	2.67	4.5	0.7	87.5	80.2
RIO-PS/ECN	2.66	3.4	1.0	91.0	83.5

exchanges, link utilization, and loss rates were most improved under DCN.

Moreover, the results demonstrate that our differential treatment of flows by size/rate does not lead to starvation of flows. While short flows do better, long flows are not unduly penalized under DCN. For contemporary models of Internet traffic (*i.e.*, for mixes of mice and elephants observed in recent measurements of Internet traffic), DCN marginally penalizes only a few very large flows and the aggregate of short flows do not starve long flows.

We believe these results are significant because they demonstrates that the benefits of AQM can be realized without the cost and new protocol deployment issues inherent in the use of ECN. Moreover, these results lead us to speculate that ECN is required with existing non-differential AQM designs not just because it improves AQM performance, but because ECN ameliorates a limitation in these designs. Non-differential AQM designs treat all flows identically when deciding whether to signal congestion. ECN packet markings helps end systems avoid timeouts that they may otherwise suffer due to packet losses under non-ECN AQM. This is especially true for small flows that do not have enough packets to trigger fast recovery when a packet loss occurs.

7. References

[1] B. Braden, *et al.*, *Recommendations on Queue Management and Congestion Avoidance in the Internet*, RFC 2309, April 1998.

[2] D. D. Clark and W. Fang, *Explicit Allocation of Best-Effort Packet Delivery Service*, IEEE/ACM Trans. on Networking, vol. 6(4), pp. 362-373, August 1998.

[3] C. Estan and G. Varghese, *New Directions in Traffic Measurement and Accounting*, Proc., ACM SIGCOMM 2002, Aug. 2002, pp. 323-336.

[4] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, *Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness*, Proc., IEEE INFOCOM 2001, April 2001, pp. 1520-1529.

[5] S. Floyd, *Congestion Control Principles*, RFC 2914, September 2000.

[6] L. Guo and I. Matta: *The War between Mice and Elephants*, Proc., ICNP 2001, Nov. 2001, pp. 180-188.

[7] C.V. Hollot, Vishal Misra, Don Towsley, and W. Gong, *On Designing Improved Controllers for AQM Routers Supporting TCP Flows*, Proc., IEEE Infocom 2001, pp. 1726-1734.

[8] http://www.iet.unipi.it/~luigi/ip_dummynet/

[9] C. Kenjiro, *A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers*, Proc., USENIX 1998 Annual Technical Conf., New Orleans, LA, June 1998, pp. 247-258.

[10] L. Le, J. Aikat, K. Jeffay, F. D. Smith, *The Effects of Active Queue Management on Web Performance*, Proc., ACM SIGCOMM 2003, Aug. 2003, pp. 265-276.

[11] D. Lin and R. Morris, *Dynamics of Random Early Detection*, ACM SIGCOMM 97, Sept. 1997, pp. 127-137.

[12] R. Mahajan, S. Floyd, and D. Wetherall, *Controlling High-Bandwidth Flows at the Congested Routers*, Proc., ICNP 2001, pp. 192-201.

[13] T. J. Ott, T. V. Lakshman, and L. H. Wong, *SRED: Stabilized RED*, Proc., IEEE INFOCOM 1999, pp. 1346-1355.

[14] J. Padhye, and S. Floyd, *On Inferring TCP Behavior*, Proc., ACM SIGCOMM 2001, Aug. 2001, pp. 287-298.

[15] J. Padhye, and S. Floyd, *TBIT, The TCP Behavior Inference Tool*, http://www.icir.org/tbit/

[16] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, *Approximate Fairness through Differential Dropping*, ACM CCR, April 2003, pp. 23-39.

[17] R. Pan, B. Prabhakar, and K. Psounis, CHOKE, *A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation*, Proc., IEEE INFOCOM 2000, March 2000, pp. 942-951.

[18] F.D. Smith, F. Hernandez Campos, K. Jeffay, and D. Ott, *What TCP/IP Protocols Headers Can Tell Us About The Web*, Proc., ACM SIGMETRICS 2001, June 2001, pp. 245-256.

[19] I. Stoica, S. Shenker, and H. Zhang, *Core-Stateless Fair Queuing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks*, Proc., ACM SIGCOMM 1998, Aug. 1998, pp. 118-130.

[20] F. Hernandez Campos, K. Jeffay, and F.D. Smith, *Modeling and Generation of TCP Application Workloads*, Technical Report, 2004 (*in submission*).

[21] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, *On the Characteristics and Origins of Internet Flow Rates*, Proc., ACM SIGCOMM 2002, Aug. 2002, pp. 161-174.