The Power of Explicit Congestion Notification

Aleksandar Kuzmanovic
Department of Computer Science
Northwestern University
akuzma@cs.northwestern.edu

ABSTRACT

Despite the fact that Explicit Congestion Notification (ECN) demonstrated a clear potential to substantially improve network performance, recent network measurements reveal an extremely poor *usage* of this option in today's Internet. In this paper, we analyze the roots of this phenomenon and develop a set of novel incentives to encourage network providers, end-hosts, and web servers to apply ECN.

Initially, we examine a fundamental drawback of the current ECN specification, and demonstrate that the absence of ECN indications in TCP control packets can dramatically hinder system performance. While security reasons primarily prevent the usage of ECN bits in TCP SYN packets, we show that applying ECN to TCP SYN ACK packets can significantly improve system performance without introducing any novel security or stability side-effects. Our network experiments on a cluster of web servers show a dramatic performance improvement over the existing ECN specification: throughput increases by more than 40%, while the average web response-time *simultaneously* decreases by nearly an order of magnitude.

In light of the above finding, using large-scale simulations, modeling, and network experiments, we re-investigate the relevance of ECN, and provide a set of practical recommendations and insights: (i) ECN *systematically* improves the performance of all investigated AQM schemes; contrary to common belief, this particularly holds for RED. (ii) The impact of ECN is highest for web-only traffic mixes such that even a generic AQM algorithm with ECN support outperforms all non-ECN-enabled AQM schemes that we investigated. (iii) Primarily due to moderate queuing levels, the superiority of ECN over other AQM mechanisms largely holds for high-speed backbone routers, even in more general traffic scenarios. (iv) End-hosts that apply ECN can exercise the above performance benefits instantly, without waiting for the entire Internet community to support the option.

Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols

SIGCOMM'05, August 21–26, 2005, Philadelphia, Pennsylvania, USA.
Copyright 2005 ACM 1-59593-009-4/05/0008 ...$5.00.

General Terms

Algorithms, Measurement, Performance, Experimentation

Keywords

Explicit congestion notification, Congestion control, Active queue management

1. INTRODUCTION

For more than a decade, the networking research community has invested enormous efforts in the development of Active Queue Management (AQM) algorithms for the Internet, with the goal being to allow network operators to simultaneously achieve high throughput and low average delay. The key idea is to detect congestion in its early stages and signal this information to the endpoints, before the router queue overflows. In such scenarios, AQM algorithms are not forced to drop packets in order to implicitly notify endpoints about the congestion; instead, they can mark packets and send *explicit* congestion notifications to the endpoints. Such explicit indications enable much smoother end-point control [12], which in turn significantly improves system performance [12, 23]. Similar efforts are being undertaken to make both routers and endpoints in the Internet ECN-capable [30, 31].

Despite the above efforts, recent network measurements reveal an extremely poor *usage* of ECN. For example, experiments on over 84,000 web servers in the Internet indicate that in the year 2000, only 1.1% of the servers were ECN-capable [28], while this fraction increased to only 2.1% in 2004 [27]. More interestingly, measurements from [27] reveal that in experiments with ECN-enabled servers, *not a single packet* was marked by intermediate routers. This indirectly indicates that the percentage of routers that apply ECN-enabled AQM is probably even smaller than the above percentage of ECN-enabled web servers.

The causes of the above phenomenon are diverse. On one hand, deploying any change in a large scale system such as the Internet is a non-trivial engineering task. One of the reasons for the small fraction of ECN-enabled endpoints is the existence of "broken" firewalls and load-balancers in the current Internet, which incorrectly send a reset in response to a TCP SYN packet that uses ECN flags in the TCP header. While this problem has been addressed [14] and the defect has been gradually removed, this initial stress significantly reduced the ECN deployment rate because endpoints were reluctant to apply it.

On the other hand, the reasons for the small usage of AQM and ECN in the Internet routers are more serious. Despite numerous theoretical and empirical indications that AQM can indeed simultaneously improve network throughout and bound queuing delay, questions, doubts, and counter-opinions are still being expressed:

(i) Why should I drop packets when my buffers are not full [26]?; (ii) static AQM parameters cannot handle dynamic network traffic [26]; (iii) setting AQM parameters is tedious, particularly for web traffic [10]; (iv) ECN can improve performance of some AQM schemes, but not others [23]; etc. Although some of the above issues are addressed here and elsewhere [15], network providers apparently are waiting for a more uniform and stronger signal from the research community before applying any change.

In this paper, we develop a set of novel incentives for network endpoints, both web-clients and servers, to apply ECN; in addition, we develop novel incentives for network providers to apply ECN-enabled AQM schemes. We show that ECN is *not* an obstacle for AQM deployment, as suggested in [24]; moreover, the key hypothesis of our work is that ECN should be used as the driving force for AQM deployment.

In Section 2, we provide the necessary background on ECN. Next, in Section 3, we point out a fundamental drawback of the current ECN specification which drops TCP control packets in moments of congestion; we argue that marking TCP SYN ACK packets at congested routers can significantly improve the system performance without inducing any novel security or stability challenges. Section 4 evaluates the impact of this innovation on the performance of several AQM schemes in a web-browsing environment. In Section 5, we develop a simple queuing model to explain the observed system behavior. Section 6 evaluates ECN's incremental deployability, while Section 7 presents a set of experiments conducted on a cluster of web servers. We discuss related work in Section 8. Finally, in Section 9, we conclude.

2. BACKGROUND

Explicit Congestion Notification is inherently coupled with the idea of Active Queue Management. The primary goal of AQM algorithms, which we discuss in more detail below, is to allow network operators simultaneously to achieve high throughput and low average delay by detecting incipient congestion. This is achieved by sending appropriate indications to the endpoints before the queue overflows. However, the method of informing sources of congestion is not limited to dropping packets, as is the case with non-AQM-enabled FIFO queues. Instead, AQM-enabled routers can *mark* packets during congestion by setting the ECN bit in the packets' header, as originally proposed for the DECbit scheme [32]. The actual number and choice of packets that are marked during congestion depends on a particular AQM policy. The recommendations for TCP's response to ECN are initially proposed in [12], and additionally refined in [30, 31].

2.1 Negotiating ECN capabilities

Before any ECN-enabled data exchange can take place between two endpoints, they first have to successfully *negotiate* the use of ECN. ECN negotiation happens during the TCP connection setup phase. The ECN-related bits are (i) ECN-Capable (ECT) and (ii) Congestion Experienced (ECN/CE) bits in the IP header, and (iii) ECN-Echo bit in the TCP header.[1] We illustrate the negotiation procedure in Figure 1 using an HTTP file download example, which we extensively exploit later in the paper. The client first sets the ECN-Echo bit in the TCP header of a TCP SYN packet and sends this packet to the receiver. For a SYN packet, the ECN-Echo bit is defined *not* as a return indication of congestion, but instead as an indication that the sending TCP is ECN-capable [13]. Upon receiving the TCP SYN packet, the server sets the ECN-Echo bit in the SYN-ACK packet's TCP header, and sends this packet back to the client.

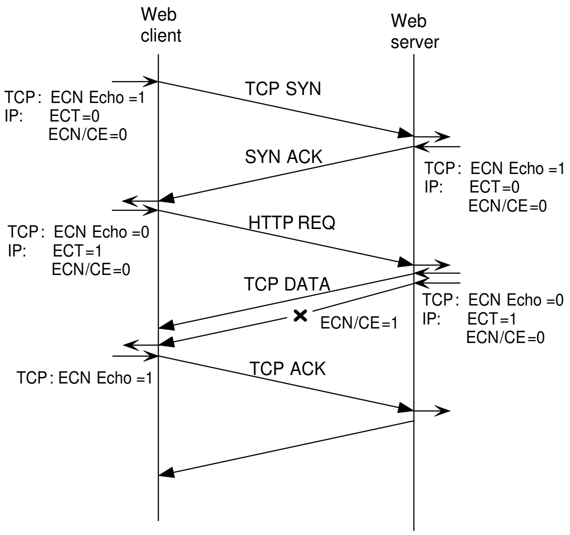

Figure 1: Negotiating ECN capabilities

When the client receives the above SYN ACK packet, the ECN capability is negotiated, and both endpoints start an ECN-capable transport by setting the ECT field in the *IP header* of data packets. In our particular scenario, the client sets the ECT bit when it sends the HTTP request. Likewise, the HTTP server also sets the ECT bit in the TCP data packet headers, when the requested file is sent to the client. In moments of congestion, which we assume happen in the direction from the server to the client, the ECN-enabled router marks ECT-enabled packets by setting the ECN/CE bit in the IP header, as illustrated in the figure. When such packets reach the client, the client sets the ECN-Echo bit in the TCP header of the corresponding ACK packet thus signaling to the server that the incoming data packet has experienced congestion.

3. ECN$^+$: ADDING ECN TO TCP'S CONTROL PACKETS

While the current ECN specification enables congested routers to mark TCP *data* packets during congestion, this is not the case with TCP control (TCP SYN and SYN ACK) packets. This is simply because these packets are used initially to negotiate the use of ECN options between the two endpoints. Below, we first elaborate on the devastating effects that this can have on system performance, particularly in AQM-enabled environments dominated by web-traffic. Then we explore possibilities of using ECN bits in the IP headers of TCP control packets. We demonstrate that marking (instead of dropping) TCP SYN ACK packets, while leaving the treatment of the initial TCP SYN packet unchanged from current practice, can only improve performance without causing a threat for system security or stability.

3.1 The Problem of TCP's "Admission Control"

Assume the scenario from Figure 1. When the client sends a TCP SYN packet, it sets a retransmission timeout timer to an initial value of 3 seconds [29]. If the client receives a SYN ACK packet before the timer expires, it sends the acknowledgement to

[1] Another TCP header's ECN-related bit, Congestion Window Reduced (CWR), is not essential for our discussion here. See RFC 3168 [31] for more details.

the server, typically piggybacking some data (a HTTP request in our scenario) with the acknowledgement. However, if the SYN ACK packet does not return (either because the TCP SYN packet is lost on the forward path, or the SYN ACK packet is lost on the reverse path) before the timer expires, the client doubles the retransmission timeout value and re-sends the TCP SYN packet. Once a SYN ACK packet is received at the client side, the connection is assumed to be successfully “admitted” into the system.

Consider first a non-AQM-based FIFO queue at the router. The key problem is that a packet loss alone is an extremely unreliable indication that the flow should not be “admitted” into the network. TCP flows are greedy and tend to utilize all possible available bandwidth. Thus, even a small number of “admitted” greedy TCP flows can create an environment with a high packet loss probability. Yet, this does *not* mean that another TCP flow cannot be admitted into the system. Moreover, TCP’s additive-increase multiplicative-decrease (AIMD) mechanism enables all flows to utilize their proportional fair share of bandwidth once they are present in the system. However, in the absence of any explicit notification from the network, a TCP endpoint has no other option but to wait for the retransmission timer to expire, and to then re-send the TCP SYN packet.

The above problem is even more serious when the congested router applies an AQM algorithm, as we demonstrate later in the paper. This is because AQM schemes employ mechanisms that drop packets *before* the queue size reaches the queue limit. While such mechanisms can have remarkable impact and can significantly improve system throughput by controlling behavior of already admitted flows [8, 16, 19, 21], they can produce devastating effects in scenarios where flows dynamically arrive and depart from the system at a high rate. This happens because the percentage of the traffic that is made up of SYN ACK packets from the server to the clients can be quite high. Not surprisingly, it has been experimentally shown that for links carrying only web traffic, AQM (e.g., RED) provides no clear advantage over drop-tail FIFO for end-user response times [10].

Unfortunately, the problem is not mitigated by ECN because ECN is *not* used in IP headers of TCP SYN or SYN ACK packets. Therefore, in moments of congestion, an ECN-enabled router *drops* TCP SYN and SYN ACK packets because the ECN option is not yet negotiated between the endpoints. Surprisingly, we demonstrate later in the paper that the corresponding performance degradation can be even worse when the AQM scheme is ECN-enabled. Below, we explore possibilities of applying ECN bits in the IP headers of TCP control packets.

3.2 Marking TCP Control Packets

The TCP “admission control” problem can potentially be alleviated by allowing endpoints to set the ECT bit in the IP headers of TCP SYN or SYN ACK packets. That would enable ECN-based AQM schemes at routers to mark TCP control packets. However, such an approach raises several concerns that we discuss below.

There are several reasons why the ECT field should *not* be set in TCP SYN packets. First, as indicated in [13], there are no guarantees that the other endpoint (web server in our scenario) is ECN-capable, or that it would be able to understand and react if the ECN/CE bits were set by a congested router. Second, the ECT field in TCP SYN packets may be misused by malicious clients to improve the well-known TCP SYN attack, where the goal is to congest the web server’s listen queue by sending a large number of TCP SYN packets. By setting the ECT bit in TCP SYN packet’s headers, a malicious client would be able to easily inject a large number of TCP SYN packets through a potentially congested ECN-enabled router. Luckily, in typical client-server scenarios (e.g., web traffic example from Figure 1), congestion is much more likely to happen in the direction of the server to the clients. Thus, setting ECT bits in TCP SYN packets is not justified from the performance point of view.

There are just as many reasons to set the ECT fields in SYN ACK packets. Refer again to Figure 1. First, when the web server receives a TCP SYN packet with the ECN-Echo bit set, it indicates that the client is ECN-capable. Hence, if the server is also ECN-capable, there are no obstacles to immediately applying ECN, and setting the ECT bit in the SYN ACK packet. Second, setting the ECT bit in SYN ACK packets does not raise *novel* security vulnerabilities. For example, provoking web servers or hosts to send SYN ACK packets to third parties in order to perform a ”SYN ACK flood” attack would be greatly inefficient. This is because the third parties would immediately drop such packets, since they would know that they didn’t generate the TCP SYN packets in the first place. Moreover, such attacks would have the same signatures as the existing TCP SYN attacks. Also, provoking web servers or hosts to reply with SYN ACK packets in order to congest a certain link would also be highly inefficient because SYN ACK packets are small in size. Such attacks would be several *orders of magnitude* weaker than the existing ICMP echo-reply DoS attacks. Finally, because the congestion is likely to happen in the server-to-client direction, setting the ECT bit in SYN ACK packets can have a tremendous impact on performance, as we indeed demonstrate below.

3.2.1 Reacting to ECN Signals in TCP Control Packets

The TCP sender should immediately send an HTTP request upon receiving a SYN ACK packet, despite the state of the ECN/CE bit. As discussed above, the fundamental reason is that the existence of a congestion notification is *not* a valid indication that the flow should not be “admitted” into the system; that is only a *necessity* when packet losses are used to convey the network state. Below, we argue that such behavior does not introduce any threat to system stability.

There are three reasons why the above behavior will *not* cause a congestion collapse. First, if the network is indeed congested, the first data packet will re-experience congestion at the router, which will set the ECN/CE bit in the first TCP DATA packet. This will force the web client to set the ECN-Echo bit in the corresponding TCP ACK packet, which will further cause the web server to initially wait for a timeout of 3 seconds before re-sending the packet,[2] and even longer if the congestion persists. Thus, the exponential backoff mechanism, which is necessary to protect network stability, is still in place. Second, AQM algorithms are able to control extremely large flow aggregates (e.g., [19]). Third, we demonstrate in Section 7 that even in an extremely heavily congested scenario caused by short-lived flows, the above approach only improves the performance without causing any stability side effects.

Finally, to distinguish the existing ECN specification from the addition proposed here, we name the above scheme ECN$^+$. In summary, while the current ECN specification enables routers to mark data packets, ECN$^+$, when enabled at servers, extends this feature to TCP SYN ACK packets. We evaluate both schemes below.

[2] Because this can cause similar performance degradations as when a SYN ACK packet is lost, RFC 2414 [6] proposes an increase of the initial window size to 2 packets in order to alleviate the above problem: if at least one of the packets returns to the sender, the connection will not suffer the 3 second -long timeout penalty.

4. THE IMPACT OF ECN$^+$ ON AQM PERFORMANCE

4.1 AQM Algorithms

While ECN$^+$ is a generic extension to ECN that should improve the performance of *all* ECN-enabled AQM schemes, we necessarily limit our performance evaluation to a subset of AQM schemes. In particular, we evaluate the impact of ECN$^+$ on Random Early Detection (RED) [15], Random Exponential Marking (REM) [8], and Proportional Integrator (PI) [19].

RED uses a weighted-average queue size as a measure of congestion, and the drop (mark) rate depends on minimum and maximum threshold parameters (denoted as min_{th} and max_{th}, respectively), as follows: when the weighted average is smaller than min_{th}, no packets are marked or dropped. When the average queue length is between min_{th} and max_{th}, the probability of marking or dropping packets varies linearly between 0 and a maximum drop probability (typically set to 0.1). If the average queue length exceeds max_{th}, all packets are marked or dropped. Reference [15] defines further refinements of the scheme.

Of particular importance is the way RED behaves when the average queue length exceeds max_{th}. The original RED paper [16] recommends marking these packets when ECN is enabled, while RFC 3168 [31] recommends dropping packets in these scenarios, even if they are ECN-enabled. The latter rule, which we analyze in more detail below, is motivated by a need to more efficiently deal with non-responsive flows that ignore congestion indications. Interestingly, we discover that both of the above implementations are represented in today's Internet. For example, Linux machines, which we use in our testbed experiments in Section 7, *mark*, by default, all packets when the average queue length exceeds max_{th}. Some other vendors follow the RFC 3168 recommendation more closely, at least according to the publicly available specifications of their equipment.[3] Because the issue of marking vs. dropping packets beyond max_{th} impacts the system performance in a non-trivial manner, we evaluate both versions below.

REM and PI apply control theoretic principles when deciding which packets to drop or mark. Both schemes measure the difference between the targeted and measured queue lengths, and increase or decrease the marking or dropping probability according to a particular control function (see references [8, 19] for details). The parameters used to set the control algorithm's targeted objective are *queue reference* (q_{ref}) in PI's case, and *target queue length* (b^*) in REM's.

4.2 Experimental Methodology

Next, we conduct a large number of large-scale ns-2 simulations. We adopt the model developed in [11], and combine it with the empirical file-size distribution reported in [33]. In this model, clients initiate sessions from randomly chosen web sites with several web pages downloaded from each web site. Each web page contains several objects, each of which requires a TCP connection for delivery. We explore the effects of persistent HTTP connections in Section 7. The inter-page time distribution is Pareto, while we generate web file sizes by fitting the empirically-measured heavy-tailed distribution reported in [23, 33]. While the majority of the flows are very short, such that the mean file size is 7.2 kBytes, Gbytes file sizes will also be generated such that the top 15% of object sizes approximately accounts for 80% of the bytes sent by servers [33]. According to [23], the combination of heavy-tailed user "think times" and the above file-size distribution creates long-range dependant (LRD) traffic with a Hurst parameter between 0.8 and 0.9. We uniformly distribute the flow round-trip times in the range from 10 ms to 150 ms.

Our simulation scenario consists of a web-client and a web-server pool that are interconnected by a pair of routers and a bottleneck link. Each node from the client pool connects to a router R1 with a 1 Gbps link; likewise, each node from the server pool connects to another router, R2, via a 1 Gbps link. Nodes R1 and R2 are connected by a link whose capacity we change from 100 Mbps to 1 Gbps. We adopt the experimental method of [23], and proceed in two steps. First, we set the capacity between R1 and R2 to 1 Gbps, and vary the number of active web sessions in the system. In this way, we place a nominal offered load on an uncongested link, in the direction from R2 to R1. We generate offered loads in the range from 80 Mbps to 105 Mbps, and we explain the reasons for such a choice below. The web response times measured in this uncongested environment represent the ideal system behavior, which we later use to evaluate the performance of various AQM schemes in congested environments.

Second, we reduce the R1-R2 capacity to 100 Mbps and re-run the above web-request traces with the goal to evaluate the performance of a particular AQM scheme implemented at R2. As experimentally evaluated in [23], and as we analytically show in Section 5, AQM schemes impact performance when the utilization exceeds 80%; hence, we explore such congestion levels. Due to space constraints, we report the results only for 90 Mbps and 105 Mbps. In the rest of the paper, we refer to the 90 Mbps load on a 100 Mbps link as the *lightly* congested scenario, whereas we refer to the 105 Mbps case as the *persistently* congested scenario.

We set the AQM parameters as follows. For RED, we set the RED's *targeted delay* parameter to 5 ms, and let the algorithm from [15] automatically set all other parameters. For REM and PI, we set b^* and q_{ref} to 62 kBytes, which corresponds to the same targeted queuing delay of 5 ms, on a 100 Mbps link. The performance measures of interest are end-to-end response times for each request/response pair, and throughput on the bottleneck link. For a given file, we compute its response time from the moment when the first request for the file is sent to the server, until that file is successfully downloaded by the client. We report the cumulative distribution function (CDF), $F(x) = Pr[X \leq x]$, of response times up to 2 seconds.

4.3 Response Times

4.3.1 *RED and RED**

Here, we evaluate the impact of ECN$^+$ on two versions of RED. The first is the version in which all packets are dropped when the average queue length exceeds max_{th}, which we denote below as RED. The second version is the one in which all packets are marked in such scenarios, which we denote as RED*.

Figure 2 depicts the CDF response-time profiles for RED without ECN, with ECN, and with ECN$^+$, when the offered load is set to 90% and 105%. As expected, the uncongested network scenario (1 Gbps link between R1 and R2) has the best response-time profile, since the percentage of successfully-transmitted files (the y-axis in Figure 2) is the largest for all given response times (the x-axis in the figure). Another expected result is that for any given scheme, the profile for 90% load is better than the corresponding profile for 105% simply because the congestion is more persistent in the latter scenario. Finally, as previously reported in [23], ECN alone provides a small improvement to the non-ECN scenario. Below, we argue that this is a direct consequence of RFC 3186's rule to drop

[3] Detailed information about implementation and deployment of ECN is available at *http://www.icir.org/floyd/ecn.html*.

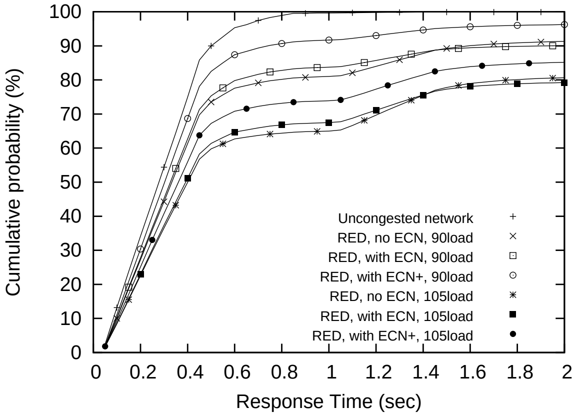

Figure 2: RED performance

all packets when the average queue length exceeds max_{th}.

The key insights from Figure 2 are the following. First, note that ECN$^+$ indeed significantly improves the performance of RED. This is because the SYN ACK packets are marked in the ECN$^+$ case, and not dropped, as in the ECN case. Thus, a number of unnecessary timeouts are avoided, and the performance improvements are evident in the figure, both for 90% and 105% loads. However, because all packets, including SYN ACKs, are *dropped* when the RED's average queue length exceeds max_{th}, this significantly worsens the RED's response-time profile.

RFC 3168 motivates this rule with a need to more efficiently deal with non-responsive flows that are ignoring congestion indications. However, dropping *all* packets beyond max_{th} cannot protect against non-responsive flows. Instead, it can actually aid a potentially malicious user. This is because the proposed rule degrades *all* flows that share the bottleneck, not just the non-responsive ones. More sophisticated mechanisms, such as the one proposed in [25], are needed to first detect non-responsive flows, and then drop packets exclusively from these flows.

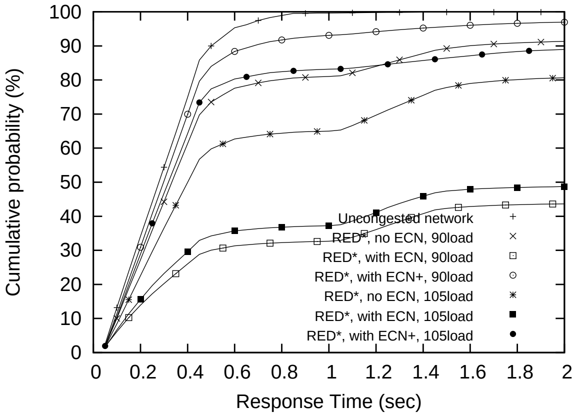

Figure 3: RED* performance

Figure 3 depicts the performance of RED* in the same simulation scenarios as above. The most stunning result is certainly the huge *degradation* of response times in scenarios with ECN (when TCP data packets are ECN-capable, but SYN ACKs are not). For example, for 90% load, the figure shows that approximately only 30% of the flows have response times less than 0.5 sec. This is a significant degradation from the scenario in which ECN is not used, where nearly 75% flows have response times less than 0.5 sec. This is due to the "TCP admission problem" discussed above; we provide additional insights below.

The only difference between the RED and RED* schemes is the way in which packets are treated when the average queue length exceeds max_{th}: they are dropped by RED, and marked by RED*. Because data packets are marked by RED*, TCP's end-point control becomes less responsive [12], and RED*'s operating point (average queuing length) moves closer to the upper threshold max_{th}. While this can increase the throughput of ECN-enabled data packets, it can have a devastating effect on non-ECN-enabled SYN ACK packets that are being frequently generated by web servers in response to client's TCP SYN packets. Because SYN ACK packets are now much more frequently dropped, the timeout penalty is invoked more often, and the degradation becomes huge. ECN$^+$ solves this problem because web servers in this scenario send ECN-enabled SYN ACK packets that are marked by the congested router. Thus, ECN$^+$ avoids the above degradation, and Figure 3 shows that it significantly improves system performance when compared to the scenario without ECN. Moreover, in the 90% load scenario, RED*'s profile with ECN$^+$ comes very close to the idealized uncongested profile.

4.3.2 REM and PI

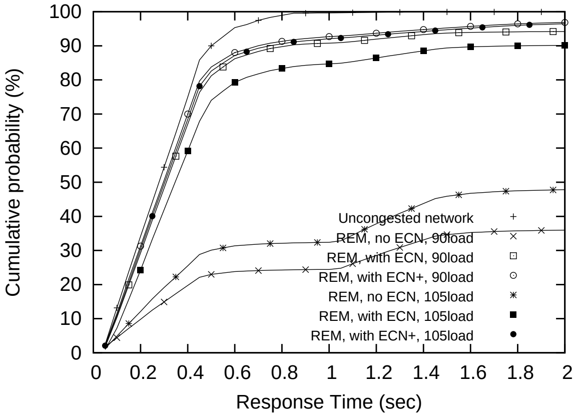

Figure 4: REM performance

Figures 4 and 5 show the impact of ECN$^+$ on REM and PI, in repeated scenarios from above. The key insight from Figure 4 is very low performance of REM without ECN support. However, note that ECN alone can significantly improve REM's performance, while the addition of ECN$^+$ has variable impact. In the 90% load scenario, ECN$^+$ only marginally improves REM's performance with ECN, which indicates that REM's marking is quite conservative in lightly congested scenarios. We analyze such scenarios in more depth in the following section. However, in the 105% load scenario, the benefits of ECN$^+$ become more pronounced, and the appropriate delay characteristic remains almost the same as when the congestion is not as persistent. Generally, when the level of congestion increases, the benefits of ECN$^+$ are more pronounced. This result systematically holds for all schemes explored in this paper. The key reason for this is that dropping SYN ACK packets on persistently congested links can significantly degrade system performance; therefore, ensuring that those packets are marked prevents the above degradation.

Figure 5 depicts the CDF response-time profiles with PI. While

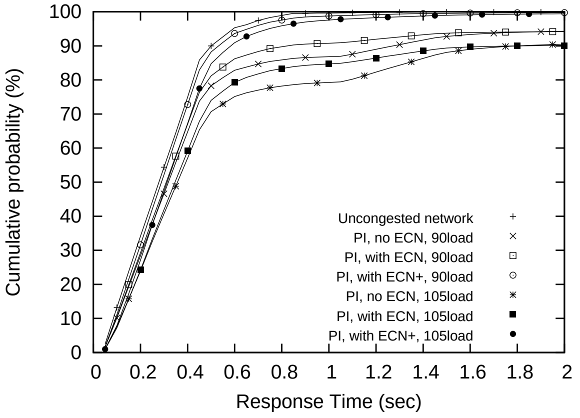

Figure 5: PI performance

ECN does improve the performance, note that the impact of ECN$^+$ is even more profound. Moreover, for both levels of congestion, the clients' response times are very close to the uncongested scenario. Because we treat PI in more detail in the following section, we now turn our discussion to another important issue, the impact of ECN$^+$ on throughput at the congested router.

4.4 Throughput

The primary objective of ECN$^+$ is to address TCP's "admission control" problem, where the loss of TCP control packets can severely impact the system performance in highly dynamic environments. While the impact of ECN$^+$ on end-to-end delay is indeed significant in the presence of *all* AQM schemes, we demonstrate below that the impact on throughput can also be surprisingly high.

Table 1 summarizes the throughput results for ECN$^+$ and ECN for all AQM schemes. The improvements of ECN$^+$ over ECN are more moderate for PI, RED, and REM, where they vary from 1% to 5%. This is somewhat expected, because ECN$^+$ impacts mostly short-lived flows that in turn cannot impact throughput considerably. Nevertheless, it is important to note that the impact on throughput is systematically positive, which means that ECN$^+$ does not improve the end-to-end response-time characteristic by degrading throughput. Instead, ECN$^+$ exploits opportunities thus far unexploited by AQM schemes.

However, in the RED* scenario, the impact of ECN$^+$ on throughput becomes quite substantial, and ranges from 6%, in the light congestion scenario, to 20% in the persistent one. The key reasons for throughput degradation in the RED*/ECN scenario are the same as for the response-time degradation. In summary, because TCP *data* packets are marked beyond max_{th}, the RED*'s operating point moves closer to max_{th}, which further causes a significant degradation for short flows as SYN ACK packets are often dropped. Unfortunately, the same happens to larger flows that are forced to wait a long time before being "admitted" into the network, which causes significant throughput degradation.

4.5 Comparing Different Schemes

While the relationship among different schemes is beyond the scope of this paper (see references [20, 23, 24] for more rigorous comparisons of various AQM schemes, as well as FIFO), we do it because the impact of ECN$^+$, while systematically positive, is non-uniform for the evaluated AQM schemes. Due to space constraints, we do not show the response-time comparisons for different AQM schemes with ECN$^+$ in a separate figure. In summary, while PI has the best performance, the difference between PI and other schemes is significantly reduced in the presence of ECN$^+$. Also, RED*'s profile is almost identical to REM's, while RED* outperforms RED. This is because ECN$^+$ improves RED*'s performance the most.

5. UNDERSTANDING ECN$^+$

5.1 Decoupling ECN$^+$ from AQM

ECN$^+$ is inherently coupled with AQM. However, while the performance of AQM schemes with and without ECN has been explored, and while the impact of ECN$^+$ on AQM performance is evidently positive, the question is: can ECN$^+$ be decoupled from AQM-specific mechanisms? In other words, our goal is to isolate the impact of ECN$^+$ from sophisticated mechanisms that define the way packets are dropped or marked at the queue. Reasons for conducting such evaluations are the following: (i) to emphasize the importance of ECN$^+$, (ii) to understand the impact of non-ECN-related AQM mechanisms on end-to-end performance, and (iii) to compare the impacts of the two in various scenarios.

To decouple ECN$^+$ from specific AQM dropping/marking mechanisms we proceed as follows. We explore a simple threshold-based AQM algorithm, which is defined as follows: when the *temporal* queue length is smaller than a given queue threshold, no packets are marked; whenever the queue length exceeds the threshold, all packets are marked. This scheme intentionally lacks all fundamental AQM mechanisms: first, it does not use the *averaged* queue length as an indication of congestion, which is needed to protect from prematurely sending congestion indications to the endpoints [16]; second, it has a sharp "step" marking function; therefore, it lacks any randomization properties and is prone to possible flow-synchronization effects that can cause significant throughput degradations [16]; finally, the threshold scheme lacks sophisticated control-theoretic mechanisms (e.g., the ones proposed in [8, 19]). However, the scheme uses ECN$^+$, which initializes smooth ECN-based endpoint control defined in [12], and enables marking of SYN ACK packets. Thus, the system's performance depends solely upon these two mechanisms.

To isolate "classical" AQM mechanisms from ECN$^+$, we compare the above scheme against dropping PI. Dropping PI posseses all the features that the above scheme lacks, yet PI in this scenario lacks the support of ECN$^+$. It is important to understand that we neither suggest that PI should not use ECN$^+$ nor that one should apply the threshold scheme. Our goal is to evaluate the impact of the two mechanisms. While necessarily not comprehensive, the experiments and analysis below provide valuable insights that are of practical importance.

5.2 Web Traffic Mixes

5.2.1 Lightly Congested Links

AQM algorithms are designed to control delay and throughput in *persistently* congested scenarios by marking/dropping packets in an effort to stabilize the queue length at a targeted level. However, in lightly congested scenarios, both classical randomization mechanisms and sophisticated control theoretic mechanisms may be of limited importance. This is because the temporal queue length may only occasionally exceed the level targeted by AQM. Thus, trying to stabilize the queue length in such scenarios may be less relevant, because the queuing oscillations are largely independent of the actual AQM mechanisms. On the contrary, the use of ECN$^+$ (i.e.,

Table 1: Normalized Throughput (%)

AQM scheme	RED/ECN	RED/ECN$^+$	RED*/ECN	RED*/ECN$^+$	REM/ECN	REM/ECN$^+$	PI/ECN	PI/ECN$^+$
90% load	84.91	85.11	73.24	79.29	78.28	78.63	86.37	86.56
105% load	94.65	95.02	76.62	96.51	94.42	99.73	99.76	99.89

marking instead of dropping packets) during these short congestion periods can have a dominant impact on end-to-end performance.

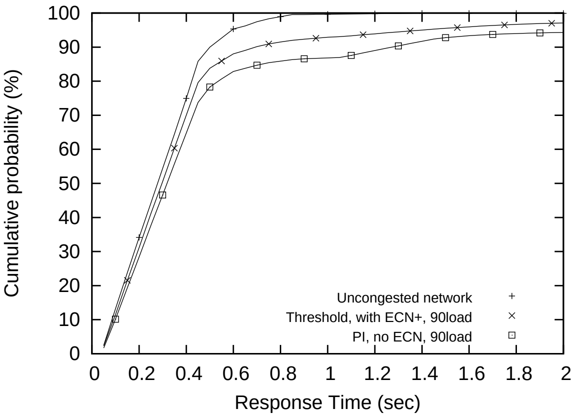

Figure 6: Threshold with ECN$^+$ vs. PI without ECN; 90% load

Indeed, Figure 6 confirms our hypothesis. It depicts the web response-time profiles of the threshold scheme with ECN$^+$ and PI without ECN. Despite the lack of sophisticated mechanisms, the threshold scheme with ECN$^+$ has a better response-time profile than the dropping PI. More surprisingly, the throughputs of the two schemes are approximately the same; 85.05% for the threshold scheme and 86.09% for PI. Below, we develop a model to further explain these results.

5.2.2 Modeling Queuing Behavior

Here, we develop a simple, yet insightful, model to understand the impact of AQM mechanisms on delay and throughput in lightly congested scenarios. While interactions between a particular AQM scheme at the router and TCP congestion control at the endpoints are essential for system operability, we make no attempts to model these complex interactions (see references [7, 17] for such attempts). Instead, we apply an *indirect* approach. We first determine the queuing behavior, in absence of AQM, as a function of link utilization. Then, we compute the probability that the queue length exceeds the level typically targeted by AQM algorithms. A large probability of exceeding the targeted queue level implies a potentially high impact of AQM mechanisms that aim to stabilize the queue length at that level, while a small probability indicates a limited impact of such mechanisms on performance.

We model the router buffer as an M$^{[X]}$/M/1 queue with a FIFO service discipline. Packets arrive in the queue in bursts of varying size governed by random variable X. The distribution of X is determined by TCP's slow-start mechanisms, the size and distribution of TCP's *receiver advertised window* parameter W_{max},[4] and the flow size distribution. Assume first a short flow of size s that never exits the slow-start phase such that its window size never reaches the bound determined by W_{max}. Thus, because the initial window size is two packets [6], and because TCP's slow-start mechanism doubles the window size each round-trip-time, the flow arrives into the system in n bursts of size $X_s = \{2, \ldots, 2^{n-1}, R_s\}$, where $R_s = s \, mod(2^n - 1)$. On the contrary, larger flows will necessarily hit the limit imposed by the receiver. Denote by l the ("large") file size in this scenario; the file arrives into the queue in bursts of size $X_l = \{1, 2, \ldots, W_{max}, \ldots, W_{max}, R_l\}$, where the actual number of bursts and the remainder factor R_l are functions of l and W_{max}. Finally, by mapping the file-size distribution using the above file-to-burst size transformations, and by using the three-modal distribution for W_{max} [27], we can compute the burst-size distribution X for any given flow-size distribution.

We justify the choice of the M$^{[X]}$/M/1 model as follows. First, the arrival process is Poisson because this realistically models high-aggregation regimes in which bursts from many flows arrive at the queue. The same argument justifies the assumption of uncorrelated burst *sizes*: bursts produced by very long flows are limited by the W_{max} parameter, and correlation among such bursts diminishes due to large numbers of other bursts that originate from many different sources. Second, the model assumes the Poisson service rate. While the service rate (packets/sec) is clearly deterministic in practice, the Poisson assumption significantly simplifies our analysis here and at the same time only moderately overestimates the queue length [17]. Finally, we do not model the impact of other bottlenecks that can exist on an end-to-end path. Any distortion of packet bursts on secondary bottlenecks would necessarily lead to even shorter queue lengths than modeled here.

Denote by ρ the load on the link, and by $E(X)$ and $E(X^2)$ the first and second moments of the burst size. Then, it could be shown that the expected queue length, $E(Q)$, can be expressed as

$$E(Q) = \frac{\rho}{1-\rho} \frac{E(X) + E(X^2)}{2E(X)}. \qquad (1)$$

The derivation is given in [22].

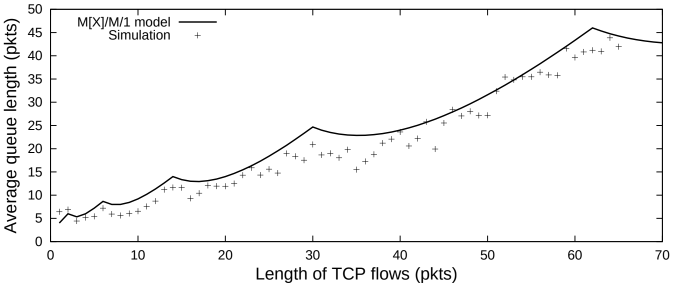

Figure 7: The average queue length as a function of the flow length for $\rho = 0.8$

Figure 7 shows the expected queue length as a function of the flow size, for a fixed load, and in a scenario where all generated flows are of the *same size*. While not representative of an actual scenario, our goal here is to illustrate a good match between the

[4] According to measurements from [27], approximately 20% of TCP flows have the advertised window parameter set to 8 kBytes, 35% to 16 kBytes, and the rest of 45% to 64 kBytes.

model and simulations. The non-monotonic relationship between the average queue length and the TCP flow length arises because the average queue length peaks when the probability of large bursts is highest and not when the average burst is highest. Our results here line up well with the ones reported in [7], which are obtained using the M/G/1 queuing model.

The key insight from Figure 7 is a particularly moderate level of queuing with respect to the queuing delay typically targeted by AQM schemes [15], despite a relatively high utilization level. This implies a limited impact of AQM mechanisms that aim to stabilize the queue length at the targeted level; an AQM algorithm can achieve this goal in a persistently congested scenario by sending more frequent congestion indications, yet, an AQM cannot increase the queue length in moments of traffic starvation. Indeed, the mean queuing delay in lightly congested scenarios may often be *below* the level targeted by AQM schemes (e.g., 5 ms, as proposed in [15]). For example, Figure 7 indicates that a typical web-browsing aggregate (e.g., the mean flow-size equals 7.22 packets [33]) would have only a moderate (5 packets) average queue length.

While quite insightful, Figure 7 does not correspond to a realistic flow-size distribution. In addition, Equation (1) computes the *average* queue length, which can be quite misleading in the case of non-standard queuing distributions. Below, we use our model in order to evaluate the impact of a realistic flow-size distribution, and also to numerically compute the corresponding queue-size distribution.

We numerically solve the system of linear equations defined by the matrix of M$^{[X]}$/M/1 transition probabilities (see [22] for details) as follows. We start from the file-size distribution used in the previous section, which is initially obtained from representative web-based network measurements [23, 33]. Next, using the file-to-burst transformations developed above, we obtain the appropriate burst-size distribution, which enables us to solve the system of M$^{[X]}$/M/1 equations and obtain the queue size distribution. Finally, we compute the probability of the queue length exceeding the level typically targeted by AQM algorithms, and present the results in Figure 8.

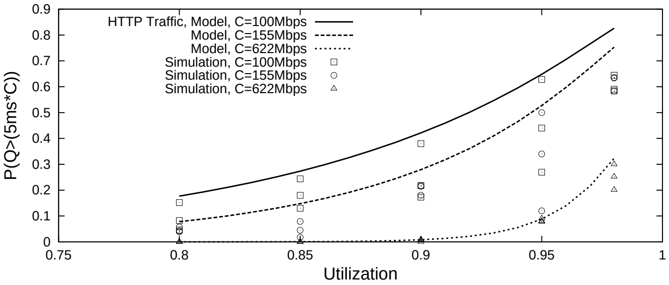

Figure 8: Probability that the queue delay exceeds 5 ms as a function of the link load (web-traffic scenario)

We denote the link capacity by C. Figure 8 depicts both modeling and simulation results for the probability that the queue length exceeds the 5 ms$*C$ level. Figure 8 depicts this probability as a function of the link utilization ρ, and for the link capacities of 100 Mbps, 155 Mbps, and 622 Mbps. In addition, we perform simulations on a *FIFO* queue for three different random seeds. Figure 8 shows a good match between modeling and simulations, with the difference that in this scenario the model behaves as an upper bound for the simulation results.

The key point from Figure 8 is that the probability that the queue length exceeds the 5 ms$*C$ threshold is indeed very small, which, based on the above discussion, indicates a similar impact of non-ECN-based AQM mechanisms. As expected, this impact increases for higher utilization levels, and decreases for higher link-speeds. For example, Figure 8 shows that for $C = 622$ Mbps and $\rho = 0.95$, the probability that the queue length exceeds the 5 ms$*C$ threshold is smaller than 10%, indicating that the corresponding congestion epochs are indeed very short. Nevertheless, AQM mechanisms are still needed to control delay during these epochs, because a simple FIFO queue lacks any such capabilities [23]. However, as indicated in Figure 6, ECN-originated mechanisms, and much less sophisticated AQM control mechanisms, are responsible for end-to-end performance. Moreover, the use of ECN$^+$ is of particular importance here, because it prevents unnecessary performance degradations (e.g., dropping SYN ACK packets) during short-lived congestion periods.

5.2.3 *Persistently Congested Links*

Here, we increase the load to 105%. This means that the population of web clients in this scenario increases such that they would generate a load of 105 Mbps on a 1 Gbps link. Therefore, this creates a persistently-congested environment for a 100 Mbps link. We show that the impact of ECN$^+$ on web response times increases in such scenarios, while the two schemes (threshold-based and PI) have approximately the same throughput. Below, we explain the origins of such a behavior.

Our results (not shown due to space constraints, see reference [22] for more details) reveal that the threshold-based scheme with ECN$^+$ outperforms PI without ECN by even a larger margin than in the above lightly-congested scenario. This is because marking, instead of dropping packets in this scenario has an even larger impact on end-to-end performance. This is particularly true for SYN ACK packets, which are marked in the case of ECN$^+$. However, a more interesting result is the impact of both schemes on normalized throughput. It is 99.89% in the PI case, while it is 97.37% for the ECN$^+$-enabled threshold scheme. While PI's control mechanisms are indeed developed for, and obviously perform well in, persistently-congested scenarios, the surprising result is the high throughput achieved by the threshold-based scheme. This is despite the fact that it lacks both generic anti-randomization mechanisms as well as more advanced control mechanisms. Below, we explain this phenomenon in more detail.

The key reasons for the high throughput achieved by the threshold-based scheme with ECN$^+$ are the following. First, while *dropping* all packets when the instantaneous queue length exceeds a given threshold can have devastating effects on TCP's performance, this is not necessarily the case when ECN is supported. This is because ECN-enabled TCP endpoints react to the event of multiple marked packets within an RTT the same as if a single packet was dropped [30]. Thus, the impact on throughput is not dramatic. Second, even though short flows carry only 20% of the bytes in our scenario, the fact that SYN ACK packets are not dropped has positive impact on throughput. However, the key reason for the good performance of this generic scheme is an obvious lack of synchronization among longer-lived flows.

Synchronization of TCP flows was one of the motivations for RED [16]. The main goal of RED is avoiding the synchronization of many TCP flows that decrease their window at the same time, and thus degrade the system throughput. The key reasons for the absence of synchronization in our scenario, despite the lack of randomization mechanisms, are the following. First, while we do generate long flows in our simulation (according to the file size distribution reported in [23, 33]), these flows are of *finite* size. Thus, they are downloaded in finite time, which can sometimes not be long enough to allow synchronization. Second, the fact that TCP flows are limited by W_{max} additionally decreases the probability that synchronization will arise. Next, heterogeneous round-trip times may also weaken these effects. Finally, in large aggregation regimes, non-synchronized greedy short-RTT TCP flows are able to quickly fill in "gaps" induced by possibly synchronized TCP sub-aggregates.

5.3 General Traffic Mixes

So far, both modeling and simulation results are based on the trace from [23, 33], which accurately represents web-traffic scenarios. Here, we extend our analysis to general traffic mixes, which are not limited to only web traffic.

We make a brief survey of recently reported measurements of general flow-size distributions, and find two such representatives. The first is reported by Garetto *et al.* in [17]; the distribution is obtained from measurements taken on an access link of a campus network; the second distribution is reported by Campos *et al.* in [9]; it is obtained from measurements on an OC-48 link between Indianapolis and Cleveland, and the trace is publicly available at *http://pma.nlan r.net/Traces/long/ipls1.html*. While both distributions have "heavier" tails than the above web-based distribution, such that the percentage of bytes that belong to long-lived flows becomes larger, only the second trace (from the OC-48 link) reveals somewhat different trends for the impact of ECN$^+$ and non-ECN-based AQM mechanisms than reported above. Below, we present those results, both for lightly and persistently congested scenarios.

5.3.1 Lightly Congested Links

Here, we repeat the simulations for the lightly congested scenario by using the file-size distribution obtained from the above OC-48 trace. It is important to understand that we do not simply plug the trace into our simulator. Instead, we use the file-size distribution which corresponds to this trace, and generate inter-arrival times in simulations to achieve 90% load on a 100 Mbps link.

The response-time profiles (not shown due to space constraints) for the two AQM schemes are similar to that of Figure 6, which confirms the dominant impact of ECN$^+$ on web response-time performance. This is because the majority of *flows* in the experiment are still short-lived, even though long flows carry approximately 90% of the bytes in this scenario. Hence, a heavier flow-size-distribution tail has no impact on web response-time performance. On the contrary, due to lack of any randomization or any other control mechanisms, the throughput of the threshold-based AQM starts to lag behind PI's more rapidly: it is 76.81% in the threshold-based AQM case, and 80.92% for PI.

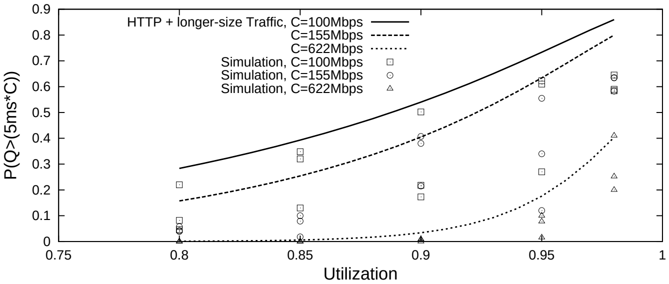

Figure 9: Probability that the queue delay exceeds 5 ms as a function of the link load (general traffic scenario)

To further understand the above behavior, we re-apply our modeling procedure and obtain the queue-size distribution that corresponds to the above general file-size distribution. Figure 9 depicts the probability that the queue length exceeds the 5 ms$*C$ threshold typically used in AQM algorithms. The *y-axis* in Figure 9 indirectly measures the "relevance" of non-ECN-based AQM mechanisms (PI's in this scenario). When compared to Figure 8, Figure 9 indicates longer queuing lengths, particularly for the 100 Mbps and 155 Mbps scenarios. For example, for 90% load on a 100 Mbps link (exactly our scenario here), the probability that the queue length exceeds the targeted AQM threshold is larger than 0.5. This indicates more persistent congestion levels, which invoke PI's control mechanisms.

On the contrary, threshold-based AQM, despite ECN$^+$ support, lacks basic control mechanisms, and experiences moderate throughput degradations. While it is well known that non-ECN-based AQM control mechanisms are required to achieve high throughput in persistently congested environments dominated by long-lived traffic flows, our results indicate that such mechanisms are required even for more moderate congestion levels. However, as the link speed increases, Figure 9 shows that despite high utilization levels, the queuing lengths are not as persistent. For example, for $C =$ 622 Mbps and 95% utilization, the queuing lengths are light, while for 90% they are almost non-existent despite heavier file-size-distribution tail. Thus, our previous analysis indicates that the generic ECN$^+$ scheme would work well in such scenarios.

5.3.2 Persistently Congested Links

Finally, we re-create the persistently congested scenario with 105% load on a 100 Mbps link, with the same flow-size distribution as above. The response-time profile (not shown due to space constraints) again confirms the dominant impact of ECN$^+$ on end-to-end performance. However, the threshold-based scheme does not keep pace with PI in throughput: threshold-based AQM has a normalized throughput of 88.43%, whereas PI achieves 96.48%. As discussed above, a larger percentage of long-lived flows increases the probability of flow-synchronization, which in turn causes throughput degradation.

6. INCREMENTAL DEPLOYABILITY

In this section, we treat the problem of incrementally deploying ECN in the Internet. Given that it is impossible to force the entire Internet community to simultaneously apply ECN, the question is how ECN- and non-ECN-enabled traffic streams affect each other when they are multiplexed. To the best of our knowledge, this issue has not yet been explored. The key problem with adding any new functionality in the Internet is to fulfill the two following, often contradictory, requirements: (i) to be "friendly" to the endpoints that do not apply the innovation; and concurrently (ii) achieve performance improvements, which are necessary to provide a reasonable incentive for endpoints to apply the innovation in the first place. While it is well-known that ECN achieves the first feature, we show below that ECN$^+$ (implemented at servers) successfully adds the second.

To become effective, ECN needs to be applied at clients, servers, and the bottleneck router in between. Below, we assume ECN support at the congestion router and ECN$^+$ support at servers, and we control the percentage of ECN flows at the router by changing the number of ECN-enabled clients. The same proportion of ECN flows in the system (and the same effects as reported below) could be achieved by assuming ECN support at clients and the congested router, and then varying the percentage of ECN$^+$-enabled servers.

Figure 10 depicts the response-time profiles for different levels of ECN deployability in the web-based simulation scenario with server and client pools. We set all the machines in the server-pool

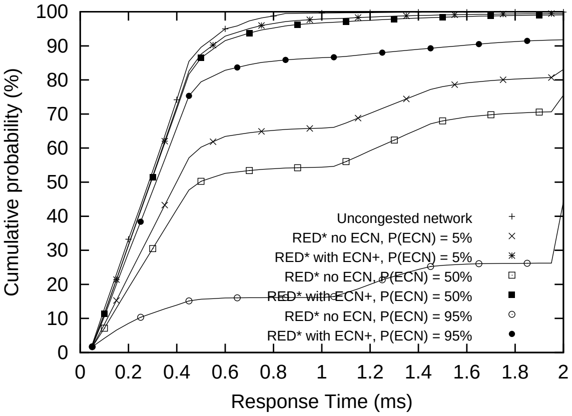

Figure 10: Incremental deployability, 98% load

to support ECN$^+$ and initially only 5% of the clients support ECN. Figure 10 shows that even the small percentage of ECN-enabled clients manage to significantly improve their response times. This is of particular importance because it provides a reasonable incentive for clients to apply ECN; by doing so, they can achieve significant performance improvements instantly, without waiting for other clients to support the option.

Next, we increase the percentage of ECN-enabled clients to 50%. Figure 10 shows that ECN-enabled clients still achieve nearly ideal performance. At the same time, the performance of non-ECN-enabled clients slightly degrades when compared to the previous scenario. This degradation occurs because a larger percentage of ECN-enabled flows better utilize the available bandwidth in this scenario and keep the average queuing length closer to RED's max_{th} parameter; this causes a larger number of SYN ACK packets belonging to non-ECN-enabled flows to be more frequently dropped at the router. However, Figure 10 indicates that the degradation is not significant. Thus, while the performance improvements are *instant* for the clients that apply ECN, the degradation of non-ECN-enabled clients is *gradual*, which is a desirable property that we discuss in more detail below.

Finally, we increase the percentage of ECN-enabled clients to 95%. The response-time profile of such clients is slightly degraded when compared to the previous scenario; this is because the system throughput increases in scenarios with high ECN deployment, as we discuss in detail in the following section. In addition, the degradation of the small number (5%) of non-ECN-enabled flows is now more pronounced. This is because such flows experience the "TCP admission control" problem (explained in detail in Section 3.1) which they can solve by applying ECN.

7. TESTBED EXPERIMENTS

Here, we perform a set of testbed experiments with the goal of verifying the above findings in a real system. The testbed consists of a cluster of Intel Pentium IV 2.0 GHz machines running Linux 2.4.18-14, with 512 MB SDRAM, and a 30 GB ATA-66 disk drive. One of the machines is configured as a router and runs Nistnet [2], an IP-layer network emulation package. The router separates the remaining machines into client and server pools. We use Nistnet to vary the RTT between clients and servers in the range from 10 to 150 ms in order to emulate a wide-area network environment. In addition, we limit the bandwidth between the two pools to 100 Mbps, which represents an uncongested scenario, and 10 Mbps, which represents a congested scenario, as we explain in detail below. This setup enables us to experiment with a version of RED implemented at the router. As explained in Section 4.1, this version, which is "hardwired" to the Linux kernel and that we denote by RED*, marks all ECN-enabled packets when the average queue length exceeds the max_{th} parameter. We set all of RED*'s parameters according to the recommendation from [15], where the reference-targeted queuing delay is set to 5 ms.

For our experimental workload, we utilize the TPC-W [5] benchmark to represent an e-commerce workload characterizing an online bookstore site that serves dynamic web content; hence, it requires access to a database server. Thus, the server pool in our scenario consists of a web-server and a database tier. At the web tier, we use a cluster of Apache web servers [4] and dynamic content coded using PHP scripts [3] at the application layer. Access to the 4 GB database tier is provided by a MySQL server [1]. The workload for TPC-W is generated by a client emulator which generates the requests specified in TPC-W.

At the client pool, the client emulator opens persistent HTTP connections to the web servers and sends a sequence of requests for the dynamic content. The mean time between the openings of two successive connections, together with the number of clients, defines the request arrival rate at the web-server tier. However, since each request for dynamic content can consist of several embedded queries, access to the database server may become the system bottleneck. Because we are interested in isolating and exploring network-based effects, we proceed as follows.

Initially, we limit the network capacity between the client and server pools to 100 Mbps. Next, we set the number of clients and the mean time between their arrivals such that the resulting average network throughput, in the direction of servers to clients, becomes 15 Mbps. At the same time, we verify that this request rate does not create a bottleneck at the database server. Finally, we limit the rate between the two pools to 10 Mbps, which enables us to explore the impact of RED* and ECN$^+$ on end-to-end performance.

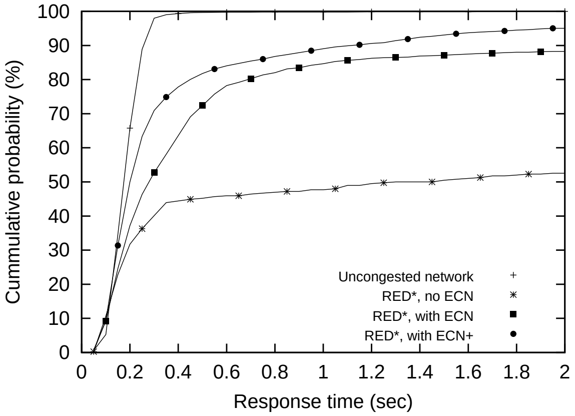

Figure 11: Testbed Experiments: CDF Profiles

Figure 11 depicts the user-experienced response-time profiles in different scenarios. The curve labeled *"uncongested network"* depicts the response times for the 100 Mbps scenario. The curve labeled *"RED*, no ECN"* depicts the response time profile in the 10 Mbps scenario, in which RED* is applied but the endpoints do not support ECN. The third curve, labeled *"RED*, with ECN,"* shows the response-time profile in the 10 Mbps scenario where we configure both client and server machines with ECN. Finally, we patch all web servers from the cluster with ECN$^+$, and label the corresponding curve *"RED*, with ECN$^+$."* While Figure 11 shows a clear improvement of ECN over the non-ECN scenario, and ECN$^+$ over the ECN scenario, the impact on throughput is even more dramatic: the normalized throughput is 44% in the scenario without ECN; 56% in the ECN scenario, and as much as 99% in the ECN$^+$ scenario. Below, we explain in detail the key reasons for such significant performance differences.

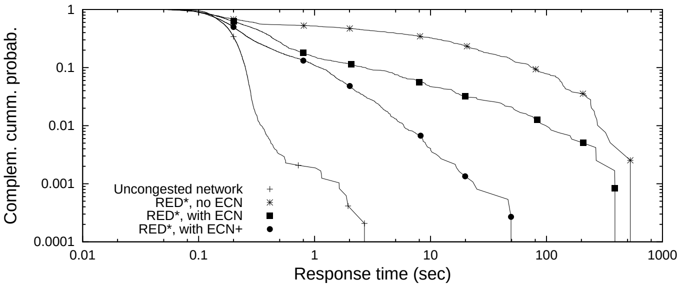

Figure 12: Testbed Experiments: CCDF Profiles

Figure 12 depicts the complementary cumulative distribution function (CCDF), $1-Pr[X \leq x]$, of response times for the above four scenarios. The smaller the tail of the distribution is, the smaller the mean response time, and the better the performance of a particular scheme. Certainly, the uncongested scenario shows superior performance. On the contrary, RED* without ECN has the heaviest tail, which indicates that a large number of web responses experience multiple successive timeouts such that the mean response time becomes 26 sec. At the same time, because most of the flows spend time in long exponential backoffs, they are unable to successfully utilize the available bandwidth; therefore, the normalized throughput is as low as 44%. Next, the presence of ECN in TCP data packets improves both the mean response time, which now becomes 4.5 sec, and the throughput, which increases to 56%. However, the key point from the figure is the large performance improvement of ECN$^+$ over ECN. In the ECN$^+$ scenario, the presence of ECN in both data and SYN ACK packets reduces the mean response time to approximately 500 ms, while the normalized throughput becomes as high as 99%. Most important, Figure 12 indicates that ECN$^+$ does not achieve performance improvements by sacrificing system stability. On the contrary, TCP endpoint control still applies exponential backoff, and some flows necessarily experience multiple timeouts due to extremely heavy congestion, as shown in Figure 12. However, despite these circumstances, ECN$^+$ avoids a large number of *unnecessary* timeouts; when compared to the existing ECN specification, ECN$^+$ shifts the system closer to an ideal operating point: web-servers manage to successfully serve approximately 50% more requests, the network throughput improves by more than 40%, and the average client-experienced response time improves by nearly an order of magnitude.

8. DISCUSSION AND RELATED WORK

ECN$^+$ extends the existing ECN specification by enabling marking, instead of dropping, server-generated TCP control packets. In this section, we discuss whether it is possible to achieve the same performance simply by giving priority to TCP control packets at routers. We also compare the ECN$^+$ approach with AQM algorithms that give preferential treatment to short flows.

The first question is whether it is possible to achieve similar effects simply by giving priority to TCP control packets at routers. Giving priority to TCP SYN packets is certainly not an acceptable option, because it opens the door to TCP SYN flood attacks. On the contrary, giving priority to SYN ACK packets at routers would certainly not have the same impact on performance. While the use of ECN in control packets certainly is important, this functionality alone is not sufficient to achieve desirable performance without an AQM algorithm at the router, the use of ECN in the TCP data plane, and the ECN-enabled end-point congestion control. In this paper, we showed that all of these mechanisms are essential to achieve improved performance.

Next, because ECN$^+$ achieves the largest performance improvements in web-based scenarios, where short flows are dominant, we compare the ECN$^+$ approach with AQM schemes and architectures that give preferential treatment to short flows. Guo and Matta [18] use different marking/dropping functions at the routers and a packet classifier at the network edge to distinguish between long- and short-lived TCP flows. While implementing such classifiers in the Internet is indeed a challenging task, we nevertheless note that ECN$^+$ is orthogonal to the above solution, and the two could be used together.

Similarly, Le *et al.* [24] propose an AQM scheme which gives a strict priority to short flows, while it applies congestion control only to long flows. The scheme distinguishes short from long flows by tracking the number of packets that have been seen recently from each flow at the router. There are several drawbacks of such an approach. First, giving a strict priority to short flows invokes a fundamental vulnerability to malicious clients that can chop their files into small pieces in order to improve performance or perform a DoS attack. Second, this approach also creates the possibility of stability problems in environments that consist of only short flows (e.g., the above dynamic web content experiment). If all flows are given priority during congestion, high packet loss ratios are generated, causing end-to-end delay characteristic to degrade.

Finally, while we demonstrated that ECN$^+$ does not have any of the above drawbacks, we note that it also impacts a much more general set of scenarios and problems. First, it addresses a generic weakness of TCP's connection-setup mechanism in which the loss of a single control packet generates long exponential backoffs. While this is certainly of particular importance in environments dominated by short-lived flows, it also impacts the fairness among long-lived flows (not shown in the paper), because newly arriving flows can enter the system without waiting long initial timeouts. Second, ECN$^+$ is a *generic* addition to ECN functionality; its impact is not limited to any particular AQM scheme - it systematically improves all ECN-enabled AQM algorithms.

9. CONCLUSIONS

This paper re-investigated the importance of ECN in light of recent measurements that reveal an extremely poor usage of this option in today's Internet. We discovered a fundamental drawback of the current ECN specification, and showed that the use of ECN indications in TCP control packets can address an inherent weakness of TCP's handshake mechanism. A loss of a single control packet can dramatically degrade system performance, primarily due to the highly skewed distribution of Internet flow-sizes. While the use of ECN bits in TCP SYN packets can potentially reinforce a well-known server vulnerability to DoS attacks launched by malicious clients, we showed that no such obstacle exists for the use of ECN bits in server-generated control packets. Moreover, we argued that such an approach (i) is more important, because the congestion is much more likely to arise in the direction from the server to the client; (ii) does not induce a challenge for system stability, because TCP's exponential-backoff mechanisms are used; and (iii) is easy to deploy, because it requires minimal changes to servers only.

In order to deploy the above innovation at servers, and more importantly, to initiate a high-scale deployment of ECN in the Internet, we argued that it is necessary to provide a set of novel incentives that address particular needs of network providers and endpoints. On a case study of the web, we produced a set of such incentives and showed that (i) web-servers that apply the above innovation can serve approximately 50% more requests, while the average response times experienced by their clients improves by nearly an order of magnitude; (ii) ECN *systematically* improves the performance of all investigated AQM schemes (RED, REM, and PI); (iii) web clients that apply ECN can experience the above performance benefits instantly, independent of the actual number and rate at which others adopt the option.

In an attempt to fully understand the importance of ECN, enriched with the above innovation, we studied the ECN functionality in isolation from traditional randomization and control-theory-based AQM mechanisms. Our findings are as follows. (i) For web-only traffic mixes, ECN dominantly impacts web response-times due to the large number of short-lived flows, such that even a generic AQM scheme with ECN support outperforms non-ECN-enabled AQM algorithms; hence, applying ECN in such environments is a more important factor than which AQM algorithm is applied; (ii) for general traffic mixes, the superiority of ECN over other AQM mechanisms largely holds for high-speed backbone routers; this is because such traffic mixes give rise to only moderate queuing lengths in high-speed environments, despite possibly high utilization levels; (iii) for general traffic mixes at the network edge, randomization and control-theoretic mechanisms are essential to achieve high throughput; while this is a well-established result for persistently-congested scenarios, we showed that the same holds for less-persistent congestion levels.

Acknowledgments

The author is grateful to Supranamaya Ranjan (Rice University) for his help with the testbed experiments, and Michele Garetto (Rice University) for discussions about the queuing model. The author also thanks the anonymous reviewers as well as his shepherd, Bruce Davie (CISCO), whose comments helped improve this paper.

10. REFERENCES

[1] MySQL Database Server. http://www.mysql.com.

[2] NISTNET: Network Emulation Package. http://snad.ncsl.nist.gov/itg/nistnet/.

[3] PHP Scripting Language. http://www.php.net.

[4] The Apache Software Foundation. http://www.apache.org.

[5] TPC-W: Transaction Processing Council. http://www.tpc.org.

[6] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's initial window, 1998. Internet RFC 2414.

[7] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proceedings of ACM SIGCOMM '04*, Portland, Oregon, Sept. 2004.

[8] S. Athuraliya, V. Li, S. Low, and Q. Yin. REM: Active queue management. *IEEE Network*, 15(3):48–53, May 2001.

[9] F. Campos, F. Smith, and K. Jeffay. Generating realistic TCP workloads. Technical report, 2004.

[10] M. Christiansen, K. Jeffay, D. Ott, and F. Smith. Tuning RED for web traffic. *IEEE/ACM Transactions on Networking*, 9(3):249–264, 2001.

[11] A. Feldmann, A. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proceedings of ACM SIGCOMM '99*, Vancouver, British Columbia, Sept. 1999.

[12] S. Floyd. TCP and explicit congestion notification. *ACM Computer Comm. Review*, 24(5):10–23, 1994.

[13] S. Floyd. Implementing ECN in TCP, 1998. http://www.icir.org/floyd/ECN-TCP.txt.

[14] S. Floyd. Inappropriate TCP resets considered harmful, Aug. 2002. Internet RFC 3360.

[15] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: An algorithm for increasing the robustness of RED's active queue manegement. Technical report, Aug. 2001.

[16] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[17] M. Garetto and D. Towsley. Modeling, simulation and measurements of queuing delay under long-tail Internet traffic. In *Proceedings of ACM SIGMETRICS '03*, San Diego, CA, June 2003.

[18] L. Guo and I. Matta. The war between mice and elephants. In *Proceedings of IEEE ICNP '01*, Riverside, CA, Nov. 2001.

[19] C. Hollot, V. Misra, W. Gong, and D. Towsley. On designing improved controllers for AQM routers supporting TCP flows. In *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, June 2001.

[20] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of ACM SIGCOMM '02*, Pittsburgh, PA, Aug. 2002.

[21] S. Kunniyur and R. Srikant. Analysis and design of an adaptive virtual queue (AQM) algorithm for active queue manegement. *IEEE/ACM Transactions on Networking*, 12(2):286–299, 2004.

[22] A. Kuzmanovic. The power of explicit congestion notification (extended version). *Northwestern University Technical Report*, May 2005.

[23] L. Le, J. Aikat, K. Jeffay, and F. Smith. The effects of active queue management on web performance. In *Proceedings of ACM SIGCOMM '03*, Karlsruhe, Germany, Aug. 2003.

[24] L. Le, J. Aikat, K. Jeffay, and F. Smith. Differential congestion notification: Taming the elephants. In *Proceedings of IEEE ICNP '04*, Berlin, Germany, Oct. 2004.

[25] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *Proceedings of IEEE ICNP '01*, Riverside, CA, Nov. 2001.

[26] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. In *Proc. of IWQoS '99*, London, UK, 1999.

[27] A. Medina, J. Padhye, and S. Floyd. Measuring the evoluation of transport protocols in the Internet. Technical report, 2004.

[28] J. Padhye and S. Floyd. Identifying the TCP behavior of web servers. In *Proceedings of ACM SIGCOMM '01*, San Diego, CA, Aug. 2001.

[29] V. Paxson and M. Allman. Computing TCP's retransmission timer, Nov. 2000. Internet RFC 2988.

[30] K. Ramakrishnan and S. Floyd. A proposal to add explicit congestion notification to IP, Jan. 1999. Internet RFC 2481.

[31] K. Ramakrishnan and S. Floyd. The addition of explicit congestion notification to IP, Sept. 2001. Internet RFC 3168.

[32] K. Ramakrishnan and R. Jain. A binary feedback scheme for congestion avoidance in computer networks. *ACM Transactions on Comp. Sys.*, 8(2):158–181, May 1990.

[33] F. Smith, F. Campos, K. Jeffay, and D. Ott. What TCP/IP protocol headers can tell us about the web. In *Proceedings of ACM SIGMETRICS '01*, Cambridge, MA, June 2001.