The effects of Active Queue Management and Explicit Congestion Notification on DNS Traffic

Stefanos Harhalakis, Nikolaos Samaras
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
v13@acm.org, samaras@uom.gr

Vasileios Vitsas
Department of Informatics
TEI of Thessaloniki
Thessaloniki, Greece
vitsas@it.teithe.gr

***Abstract*—Active Queue Management (AQM) and Explicit Congestion Notification (ECN) are two methods that attempt to alleviate the problems of FIFO queues and of packet drops for congestion indication respectively. Random Early Detection (RED) is the most popular AQM method. The Domain Naming System (DNS) protocol is a core Internet protocol with very high impact on user experience and overall network functionality. This paper presents experimental results that show an inefficiency of RED that results in degraded performance of the DNS protocol when compared with FIFO queues. The results that are presented show 8 to 50 percentage points drop of DNS efficiency depending on network conditions.**

***Index Terms*—Active Queue Management; Domain Name System; Explicit Congestion Notification; Random Early Detection**

I. Introduction

Internet traffic traverses through intermediate routers and their interface queues which, in most cases, are First-In-First-Out (FIFO) queues or Active Queue Management (AQM) [1] queues. While FIFO queues are simplistic drop-tail queues, AQM queues are more sophisticated and attempt to proactively indicate congestion to endpoints. Random Early Detection (RED) [1] is the most common AQM method with wide vendor support.

When using AQM, packet drops are the most common approach to congestion indication. They are used to inform endpoints that congestion is imminent and that they should take immediate action to reduce their transmission rate. This approach is compatible with most transport layer layer protocols (like the Transmission Control Protocol [2]) that appropriately react to congestion indication.

Explicit Congestion Notification (ECN) [3] is an addition to the Internet Protocol [4] that can be used with AQM queues in order to reduce or even eliminate packet drops for congestion indication. When using ECN, packets that carry data from a transport layer protocol that supports ECN are marked instead of being dropped by an AQM queue, when the queue needs to proactively indicate congestion. Currently, ECN may only be used by transport layer protocols that (a) implement congestion control and (b) are appropriately extended to take advantage of ECN.

The Domain Naming System (DNS) protocol [5] is a core Internet protocol that follows the client-server architecture. In most cases, clients use the User Datagram Protocol (UDP) [6] to make a one-datagram request to a DNS server and receive a one-datagram response. Since only one datagram is transmitted per direction, the UDP-based DNS implementations cannot perform flow control. However, because of the way AQM works, UDP-based DNS requests and replies may still be dropped by AQM queues in order to proactively warn the UDP (DNS protocol) endpoints about impeding congestion. Of course, this action is futile.

This paper points out that switching to AQM queues from FIFO queues may cause problems to the DNS protocol. While FIFO queues suffer the problem of global synchronization, they drop packets in bursts allowing more packets from low-traffic protocols (like DNS) to pass. This argument is verified by experimental results that show dramatic drop to DNS efficiency. A representative subset of the experimental results, that is presented here, shows a drop from of up to 50 percentage points of DNS protocol efficiency (from 91% to 41%) under the same networking conditions.

The rest of the paper is organized as follows: section II briefly introduces the required background for AQM, RED, ECN and the DNS protocol while section III describes the actual problem. Sections IV and V present the experimental facility setup and the software that was used for the experiments. The effects of AQM on DNS traffic are presented in section VI and section VII presents the conclusions of the experiments.

II. Background

FIFO is a simplistic approach to queuing and is best viewed as a single buffer. Whenever there is available buffer space, data are queued. When the buffer space is exhausted, excess data are dropped. This method is efficient against variations of traffic rates and traffic bursts and allows the interfaces to reach their maximum transmission rate because data are always available for transmission. However, in the case of persistent congestion, FIFO queues (drop-tail queues) are considered to be inadequate [1]. Because of the way congestion control is accomplished on the Internet, FIFO queues result in global synchronization meaning that traffic rate is reduced altogether and then increased again instead of

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

being maintained constant. The reaction of congestion control mechanisms also lags behind any congestion indication because of the end-to-end delay (Round-Trip Time - RTT). This has the side effect of congestion not being addressed early enough and thus causing excess packet drops which are considered harmful both because they needlessly consume router and endpoint resources and because they slow down data transfers.

Active Queue Management (AQM) attempts to alleviate the problems of drop-tail queues by providing congestion indication to endpoints before buffer space is exhausted. Random Early Detection (RED) is the most common AQM mechanism with wide vendor adoption. In order to avoid congestion and prevent global synchronization RED probabilistically starts dropping packets whenever the average queue length reaches a soft-limit and becomes a drop-tail mechanism whenever the queue length reaches a hard limit [7]. This way RED indicates congestion when it starts to occur instead of waiting for the queue to become full. Because of the probabilistic early congestion indication, RED compensates for the end-to-end delay allowing for the indication to reach the transmitting endpoint before the queue becomes full. In contrast with FIFO queues, where packets are dropped when buffer space is exhausted, RED queues drop packets in order to indicate congestion even when there is available buffer space. This is based on the assumption that endpoints will react to congestion indication on time and will lower their transmission rates in order to avoid congestion.

Explicit Congestion Notification (ECN) is an addition to the Internet Protocol (IP) that complements AQM allowing for congestion indication without dropping packets. When ECN is used, intermediate routers mark instead of dropping packets unless they have reached their maximum queue size. This effectively reduces packet drops causing fewer retransmissions which result in better network efficiency and protocol behavior. ECN's specification dictates that only packets that carry an ECN capable upper layer protocol should be marked instead of being dropped. Those IP packets are distinguished by an ECN Capable Transport (ECT) codepoint that is set by the transmitting node using the next two bits after the DiffServ IP header field. When those packets traverse AQM-based queues they are marked with the Congestion Encountered (CE) codepoint instead of being dropped, unless there is no available buffer space. This marking is handled by the upper layer protocol at the receiver which informs the transmitter for impeding congestion. In the case of the Transmission Control Protocol (TCP) , this is accomplished by using the "ECN Echo" (ECE) flag. Non-ECT traffic is never marked and is always dropped in order to indicate impeding congestion.

ECN requires support from the transport protocol and may only be used with IP packets that carry data from such a protocol. More specifically, ECN may only be used when the transport protocol supports congestion control and is willing to take advantage of ECN (for TCP this willingness is determined at the initial three-way-handshake where ECN usage is negotiated). Since congestion control can only be achieved with multi-packet data transfers, it is practically impossible for single-packet data transfers to take advantage of ECN. This means that packets of single-packet data transfers may be dropped when they traverse AQM queues even when there is available buffer space. This is a consequence of (a) IP's inability to distinguish data flows and (b) RED's characteristic of statistically handling packets. As shown in this paper, with the adoption of ECN by transport layer protocols such as TCP, the effects of this drawback are increased because non-ECN capable packets are handled in a less privileged way than packets from an ECN-enabled data flow.

The Domain Naming System (DNS) is considered one of the most crucial Internet protocols and has very high impact on end-users and overall Internet functionality. The DNS protocol can use both the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) but most DNS queries are performed using UDP because of its low latency. Only larger DNS data transfers (e.g. zone transfers) are performed using TCP.

III. Problem description

UDP-based DNS queries consist of one request and one reply and they are carried in small UDP datagrams. Because of the nature of the query/reply mechanism (one packet per direction) which is not sufficient for congestion control, DNS traffic is not allowed to be carried with ECT marked IP packets and is being exempt from taking advantage of ECN because neither UDP or the DNS protocol support congestion control. As a result, they suffer meaningless packet drops for the purpose of congestion indication.

Based on the above, it is obvious that the rationale behind AQM (proactively provide congestion indication) does not apply to the DNS protocol because of its characteristics. IP packets that carry DNS data cannot be distinguished by just examining the IP header, and are thus subject to the exact same rules such as (e.g.) TCP's traffic. To our knowledge, there is no justification (i.e. no published work) for the discarding of DNS traffic, except from the lack of the ability to distinguish it.

This paper studies the effects on DNS traffic of the transition from FIFO queues to AQM: Because FIFO queues burstly drop packets when they become full they force any congestion-controlled traffic (like TCP traffic) to suddenly drop its transmission rate. While this is an unwanted effect known as global-synchronization it leaves the queue half-empty (i.e. non-full) for more time allowing for non-congestion-controlled traffic like DNS traffic to pass.

As it is shown, the DNS protocol becomes inefficient when combined with AQM and its performance worsens even more when the background traffic takes advantage of ECN. This performance drop has very high impact on user experience and is able to cause multi-second delays to users browsing the Internet or other Internet-wide services like e-mail. A

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

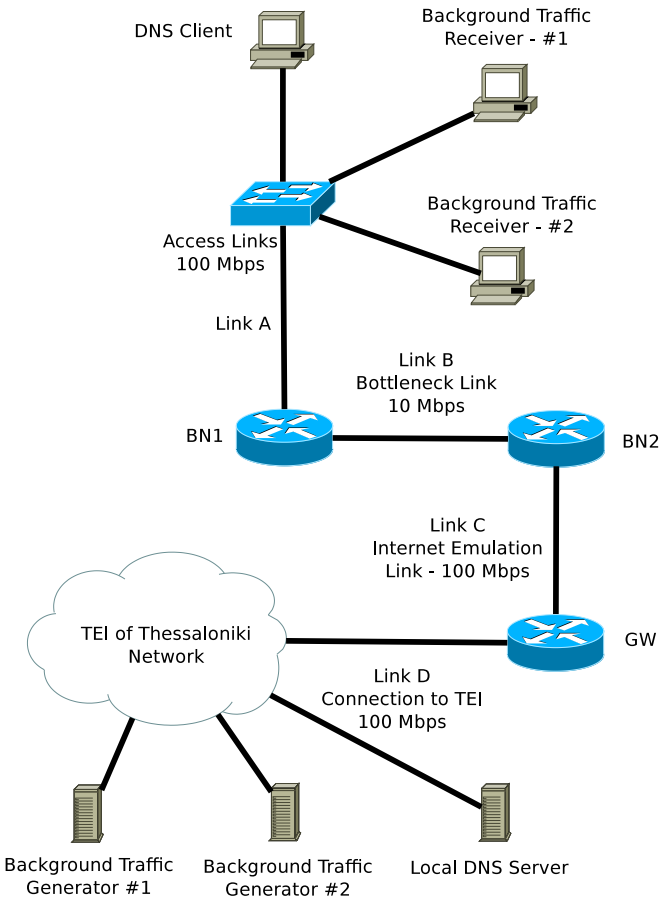

Figure 1: Testing facility setup

number of experimental results are included in order to show the performance degradation of DNS traffic when AQM is used.

IV. Facility Setup

In order to test the validity of the problem we setup an experimental facility. The setup of the facility is shown in figure 1 and consists of:

- Two servers generating constant background traffic
- Two clients receiving the background traffic
- A DNS client that performs DNS queries
- Three routers that provide the underlying network (BN1, BN2, GW)
- A 10Mbps bottleneck link with RED queues in its ends between BN1 and BN2
- A 100Mbps link that emulates Internet delays between BN2 and GW
- The local network of TEI of Thessaloniki and a 100Mbps connection to it.
- The local DNS server of the Department of Informatics of TEI of Thessaloniki

The above setup was used in order to accomplish the following goals:

- Have well-behaved background traffic: Well behaved traffic is considered all traffic that appropriately reacts to congestion indication (e.g. TCP traffic, Stream Control Transmission Protocol [8] traffic, Datagram Congestion Control Protocol [9] traffic and traffic that supports TCP-Friendly Rate Control [10]). This was accomplished by using a number of parallel TCP connections that constantly downloaded data.
- Have a bottleneck link where the data rate is limited by the hardware capabilities and not by software (i.e. traffic shaping).
- Have the ability to emulate Internet delays in order to test background TCP traffic against both low and high Bandwidth-Delay-Product (BDP) paths, since different BDP paths may result in different protocol behavior.
- Have separate clients with distinct roles. This means distinguishing traffic receivers and the DNS client. Traffic receivers only received background traffic and the DNS client only performed DNS queries. This approach eliminated any interference and removed suspicion of unwanted delays.

All experiments were performed using the following operating systems:

- Debian/GNU Linux distribution with 2.6.32 kernel for all clients and intermediate routers.
- Debian/GNU Linux distribution with 2.6.26 and 2.6.30 for the two servers.

By using the Linux operating system for intermediate routers, clients and servers we were able to measure and monitor traffic in great detail.

The administering of the interface queues is part of the traffic control (tc) facility of the Linux operating system. The actual queues are called queuing disciplines (qdiscs) and can be either classless or classful. Classful qdiscs allow the creation of multiple traffic classes and the classification of traffic. Classless qdiscs can be used either by themselves or as part of classful qdiscs where they can be attached to classes. Every network interface is required to have an attached qdisc which by default is a packet-based FIFO qdisc.

For the experiments the following qdiscs were used:

- The RED classless qdisc at the ends of link B (bottleneck link) for the AQM tests. The Linux kernel implements byte-mode RED.
- The BFIFO (byte FIFO) classless qdisc at the ends of link B for the FIFO tests.
- The Hierarchical Token Bucket (HTB) classful qdisc at the ends of link C (Internet emulation link). This classful qdisc was used in order to prioritize traffic as follows:
  - Non-DNS traffic was sent to a class that had the Network Emulator (netem) [11] classless qdisc attached in order to be delayed as needed.
  - DNS traffic was sent to a class that had a packet FIFO classless qdisc attached.

The above setup allowed for the testing of scenarios with different Bandwidth-Delay-Products (BDPs) while keeping

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

DNS measurements accurate. Since the DNS protocol doesn't do any kind of Round-Trip Time (RTT) estimation and does not perform retransmissions[1], the exemption of DNS traffic from the netem emulator did not affect the validity of the results and instead provided more accurate timings. Since link C was never congested, the usage of different qdiscs on it did not have any side effects. Remote access to the machines was performed using "out-of-band" connections that didn't traverse the bottleneck link.

While performing the tests we had to deal with a mostly undocumented feature of Linux's networking stack. While it is possible to fully customize the qdiscs of each interface, another buffer may exist at a lower level: The underlying networking driver most probably includes a buffer (referred to as ring-buffer) of its own which is large enough to affect the results. The Intel e1000 driver for example uses by default a 128 packet buffer. The size of this ring buffer in e1000 driver can be reduced to 80 packets (but not less) which still causes 90ms latency when filled. For the skge driver the minimum ring-buffer size is 48 packets resulting in 54ms latency. In order to get accurate results those value were also considered when determining the RED parameters (i.e. RED queue size thresholds were downshifted by that amount of packets).

V. Software

The experiments used the following custom-made programs:

- A client-server program that creates multiple TCP connections and constantly transfers traffic between them. The clients ran on the Background Traffic Receivers (#1 and #2) and the servers ran on the Background Traffic Generators (#1 and #2).
- A DNS resolving program that performed DNS queries as described bellow.
- A monitoring program that monitored the traffic rates and the queue status of BN2's RED queue.

All tests were performed while having 120 background TCP connections transferring data from the Background Traffic Generators to the Background Traffic Receivers. Each receiver created and maintained 30 parallel TCP connections with each server. All TCP connections used the Cubic [12] congestion control algorithm which is the default for all recent Linux kernels.

The DNS resolving program used 100 threads to perform parallel queries. Each thread performed 100 queries for warm-up and 200 queries for measurements. The warm-up allowed the network to adapt to the extra traffic and provided more accurate results. Each one of the 100*(100+200) queries was performed against a DNS name that was local to the name server in order to avoid extra (random) delay from Internet lookups. The queries were performed using the logic of the Microsoft Windows resolver which tries up to 5 times using 1, 2, 2, 4 and 8 seconds timeout respectively.

The monitoring program monitored both the interface traffic and the queue status. The interface traffic was monitored by examining /proc/net/dev 2 times per second while the queue status was monitored using the output of the "tc qdisc show dev eth1" command at a rate of 10 times per second.

The programs that were used were created as follows:

- Background traffic generator and receiver: Using C++ and the Linux's networking API.
- DNS resolver: Using the python language along with its threading support and the python-dns library.
- Monitoring program: Using the python language, the iproute2 package and the Qt library.

DNS resolving tests (i.e. with and without using ECN) did not ran in parallel in order to avoid interference with each other.

VI. Effects of AQM on DNS traffic

We evaluated the performance of the DNS protocol both when the congested link uses byte-mode FIFO queues and byte-mode RED queues. The presented results are a representative subset of a quite larger set of tests. Some of the contacted tests which used packet FIFO or more aggressive parameters showed far worse results regarding DNS timeouts.

The results presented here were achieved using the following parameters:

- Network-driver ring-buffer: 80 packets (assumed to result in 90ms delay on the 10Mbps link)
- Byte mode FIFO queue size: Depending on the targeted maximum delay. When targeting 300ms as the maximum link delay the fifo size was 262500 bytes (which results in about 210 ms maximum delay).
- RED:
  - Assumed average packet size: 1400 bytes
  - Link bandwidth: 10 Mbps
  - Targeted maximum link delay: 300ms
  - Minimum Queue Size Threshold: 74874 bytes
  - Maximum Queue Size Threshold: 262374 bytes
  - Queue Limit: 375000 bytes
  - Burst: 98 packets [2]
  - Maximum marking probability: 10%
- Internet link:
  - Delay: 100 ms
  - Delay jitter: 25 ms
  - Delay correlation: 25%

Table I shows the efficiency of the DNS protocol when FIFO and RED queues are used. The measurements were made without using ECN for the background TCP connections. As shown, the success probability of the first attempt of DNS resolvings is degraded by about 8 percentage points

[1]DNS clients may retry after predefined timeouts up to a number of times but this is a decision of the DNS client implementer. Many applications use the standard DNS resolver library of the underlying operating system. The exact DNS resolver's behavior is not dictated by the DNS protocol. DNS servers never retransmit.

[2]This is computed using the formula that is proposed by Linux's tc-red manual page: $(2 * min_q + max_q)/(3 * AvgPktSize)$. This does not have a significant effect on RED's behavior since the bottleneck link is constantly congested.

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

	FIFO queue		RED queue	
Attempt	Replied Queries	Percentage	Replied Queries	Percentage
1	19474	97.37%	17928	89.64%
2	505	2.52%	1876	9.38%
3	21	0.11%	176	0.88%
4	0	0%	17	0.09%
5	0	0%	3	0.01%
Total	20000	100%	20000	100%

Table I: Sample comparison of DNS efficiency with FIFO and RED queues

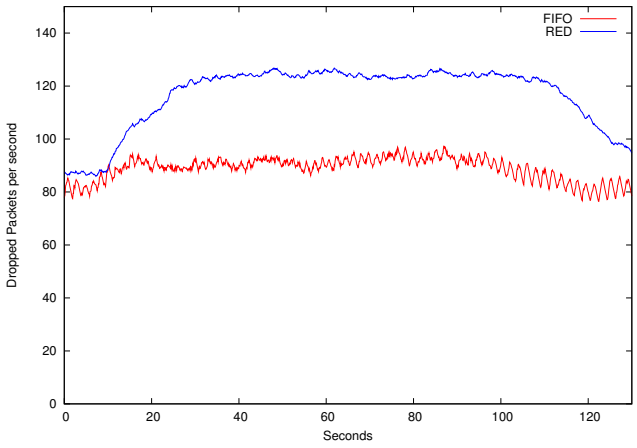

Figure 2: Packet drops for FIFO and RED queues (both DNS and non-DNS packets)

when using RED queues. When using FIFO queues the clients had 97.37% probability for getting a response while when using RED this percentage dropped to 89.64%. This is explained by the fact that RED probabilistically starts dropping packets before a queue is full meaning that there will be continuous packet drops, while FIFO queues burstly drop packets when they become full. Each burst of packet drops usually lasts a small amount of time and is followed by lower transmission rates, leaving time for DNS traffic to pass. A rough interpretation of the results indicates that when using FIFO queues the queue was not full for about 97% of the time.

Figure 2 is a graph of the average packet drop rate over time and shows the effect of DNS traffic on packet drops. The observed raise of the packet drop rate is the result of the background DNS traffic which starts at 10 seconds. Packet drops were measured 10 times per second and a rate was calculated at each timeslot (giving accuracy of 10 drops/second). The graph plots the exponential weighted moving average:

$$V_n = a * V_{n-1} + (1 - a) * P_n$$

where V_n is the value on the graph at point n, $a = 0.9$ and P_n is the measured packet drop rate (measured in 100 ms intervals and adjusted for 1 second - i.e. multiplied with 10) at point n.

The graph shows that for the time DNS traffic was generated, packet drops increased much more when using a RED queue. When using a FIFO queue, packet drops were slightly increased and the rate increment was hardly distinguished from normal traffic. With RED there was an increase of about 30% of packet drops under the exact same networking conditions. In both cases the link utilization was kept to 100%.

We further tested the DNS behavior when using AQM under the following conditions:

- With all 120 background TCP connections not using ECN. The results are those that are shown in table I.
- With half (60/120) of the background TCP connections using ECN.
- With all 120 background TCP connections using ECN.

Table II compares the results of the above test-cases and shows how the adoption of ECN by the background TCP traffic affects the DNS efficiency. In the performed tests, full ECN adoption resulted in 5 percentage points further degradation of the first attempt success probability. In the three cases the fail probability of the first attempt is about 10%, 12% and 15% respectively. For 99% confidence level the calculated confidence intervals were ±0.43, ±0.45 and ±0.63 percentage points respectively as shown in table III.

Furthermore, table IV shows the success rate of the 1st attempt of a DNS query under varying conditions. For example, when the targeted maximum delay of the bottleneck link was 200ms and the Internet delay was 20ms (±5ms) the usage of a byte-FIFO queue resulted on ~91% success rate. On the other hand, when RED was used, the success rate was:

- ~ 81% when the background traffic did not use ECN,
- ~ 41% when the background traffic was using ECN and
- ~ 68% when the background traffic was a mix of ECN and non ECN traffic.

The comparison indicates that:

1) When ECN is being used the DNS's traffic priority is practically lowered and
2) RED heavily impacts DNS traffic with relatively-small (i.e. not large) buffers

It is thus shown that AQM adoption can have harmful effects on DNS traffic, dropping the efficiency of DNS even by 10 percentage points and that ECN adoption results in even further reduction up to 50 percentage points.

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

Atmpt	no ECN	%	50% ECN	%	only ECN	%
1	17928	89.64	17535	87.67	16924	84.62
2	1876	9.38	2135	10.68	2629	13.14
3	176	0.88	282	1.41	383	1.92
4	17	0.09	46	0.23	56	0.28
5	3	0.01	2	0.01	8	0.04
Total	20000	100%	20000	100%	20000	100%

Table II: Efficiency of DNS for 300ms link delay

	Queries	Replies	Success %	Mean %	StdDev of %	Confidence Interval	Range with 99% confidence
No ECN	20000	17928	89.64	89.91	0.28618	±0.43	89.48 - 90.34
50% ECN	20000	17535	87.67	87.95	0.30139	±0.45	87.50 - 88.40
Only ECN	20000	16924	84.62	85.00	0.42665	±0.63	84.37 - 85.63

Table III: Confidence intervals with 99% confidence level from 3 samples for 300ms link delay

		FIFO queue		RED queue					
Targeted Link Delay	Internet Delay/Jitter	#	%	no ECN	%	50% ECN	%	only ECN	%
200ms	20ms ±5ms	18285	91.42	16129	80.64	13606	68.03	8278	41.39
200ms	50ms ±5ms	18513	92.56	16819	84.09	15073	75.36	9156	45.78
300ms	100ms ±25ms	19474	97.37	17928	89.64	17535	87.67	16924	84.62

Table IV: Comparison of 1st attempt success rate of DNS queries under varying parameters with FIFO and RED

VII. Conclusions

AQM and ECN are two related promising improvements for the Internet. However, as shown in this paper, the transition to AQM and ECN highly degrades the efficiency of the DNS protocol. Since UDP-based DNS queries and replies are one-packet per-direction data transfers, they cannot be congestion controlled. This results in significant DNS packet loss and has high impact on user-experience (i.e. multi-second, user-observable delays).

The paper presented a subset of a quite larger set of experimental results regarding the effects of congestion on intermediate FIFO and RED queues on the DNS protocol. Under the selected network parameters, when the congested link switched from FIFO to RED, the DNS protocol efficiency dropped from at least 8 and up to 11 percentage points. When the background TCP traffic took advantage of ECN, the DNS's efficiency further dropped from 12 up to 50 percentage points. Since the DNS protocol is a core Internet protocol with very high impact on user experience, this degradation can be considered very harmful and perhaps unacceptable especially when targeting lower maximum delays. It seems thus that the disadvantages of AQM and ECN may outweigh the advantages under certain conditions.

References

[1] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC 2309 (Informational), April 1998.

[2] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.

[3] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001.

[4] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[5] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343.

[6] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

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

[8] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol. RFC 2960 (Proposed Standard), October 2000. Obsoleted by RFC 4960, updated by RFC 3309.

[9] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340 (Proposed Standard), March 2006.

[10] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 3448 (Proposed Standard), January 2003. Obsoleted by RFC 5348.

[11] S. Hemminger. Network emulation with netem. In *Linux Conf Au*, April 2005.

[12] PFLDNet 2005. *CUBIC: A New TCP-Friendly High-Speed TCP Variant*, 2005.

978-1-4577-0681-3/11/$26.00 ©2011 IEEE