

Network Working Group
Request for Comments: 2884
Category: Informational

J. Hadi Salim
Nortel Networks
U. Ahmed
Carleton University
July 2000

Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2000). All Rights Reserved.

Abstract

This memo presents a performance study of the Explicit Congestion Notification (ECN) mechanism in the TCP/IP protocol using our implementation on the Linux Operating System. ECN is an end-to-end congestion avoidance mechanism proposed by [6] and incorporated into RFC 2481[7]. We study the behavior of ECN for both bulk and transactional transfers. Our experiments show that there is improvement in throughput over NON ECN (TCP employing any of Reno, SACK/FACK or NewReno congestion control) in the case of bulk transfers and substantial improvement for transactional transfers.

A more complete pdf version of this document is available at:
<http://www7.nortel.com:8080/CTL/ecnperf.pdf>

This memo in its current revision is missing a lot of the visual representations and experimental results found in the pdf version.

1. Introduction

In current IP networks, congestion management is left to the protocols running on top of IP. An IP router when congested simply drops packets. TCP is the dominant transport protocol today [26]. TCP infers that there is congestion in the network by detecting packet drops (RFC 2581). Congestion control algorithms [11] [15] [21] are then invoked to alleviate congestion. TCP initially sends at a higher rate (slow start) until it detects a packet loss. A packet loss is inferred by the receipt of 3 duplicate ACKs or detected by a

timeout. The sending TCP then moves into a congestion avoidance state where it carefully probes the network by sending at a slower rate (which goes up until another packet loss is detected). Traditionally a router reacts to congestion by dropping a packet in the absence of buffer space. This is referred to as Tail Drop. This method has a number of drawbacks (outlined in Section 2). These drawbacks coupled with the limitations of end-to-end congestion control have led to interest in introducing smarter congestion control mechanisms in routers. One such mechanism is Random Early Detection (RED) [9] which detects incipient congestion and implicitly signals the oversubscribing flow to slow down by dropping its packets. A RED-enabled router detects congestion before the buffer overflows, based on a running average queue size, and drops packets probabilistically before the queue actually fills up. The probability of dropping a new arriving packet increases as the average queue size increases above a low water mark $minth$, towards higher water mark $maxth$. When the average queue size exceeds $maxth$ all arriving packets are dropped.

An extension to RED is to mark the IP header instead of dropping packets (when the average queue size is between $minth$ and $maxth$; above $maxth$ arriving packets are dropped as before). Cooperating end systems would then use this as a signal that the network is congested and slow down. This is known as Explicit Congestion Notification (ECN). In this paper we study an ECN implementation on Linux for both the router and the end systems in a live network. The memo is organized as follows. In Section 2 we give an overview of queue management in routers. Section 3 gives an overview of ECN and the changes required at the router and the end hosts to support ECN. Section 4 defines the experimental testbed and the terminologies used throughout this memo. Section 5 introduces the experiments that are carried out, outlines the results and presents an analysis of the results obtained. Section 6 concludes the paper.

2. Queue Management in routers

TCP's congestion control and avoidance algorithms are necessary and powerful but are not enough to provide good service in all circumstances since they treat the network as a black box. Some sort of control is required from the routers to complement the end system congestion control mechanisms. More detailed analysis is contained in [19]. Queue management algorithms traditionally manage the length of packet queues in the router by dropping packets only when the buffer overflows. A maximum length for each queue is configured. The router will accept packets till this maximum size is exceeded, at which point it will drop incoming packets. New packets are accepted when buffer space allows. This technique is known as Tail Drop. This method has served the Internet well for years, but has the several drawbacks. Since all arriving packets (from all flows) are dropped

when the buffer overflows, this interacts badly with the congestion control mechanism of TCP. A cycle is formed with a burst of drops after the maximum queue size is exceeded, followed by a period of underutilization at the router as end systems back off. End systems then increase their windows simultaneously up to a point where a burst of drops happens again. This phenomenon is called Global Synchronization. It leads to poor link utilization and lower overall throughput [19]. Another problem with Tail Drop is that a single connection or a few flows could monopolize the queue space, in some circumstances. This results in a lock out phenomenon leading to synchronization or other timing effects [19]. Lastly, one of the major drawbacks of Tail Drop is that queues remain full for long periods of time. One of the major goals of queue management is to reduce the steady state queue size [19]. Other queue management techniques include random drop on full and drop front on full [13].

2.1. Active Queue Management

Active queue management mechanisms detect congestion before the queue overflows and provide an indication of this congestion to the end nodes [7]. With this approach TCP does not have to rely only on buffer overflow as the indication of congestion since notification happens before serious congestion occurs. One such active management technique is RED.

2.1.1. Random Early Detection

Random Early Detection (RED) [9] is a congestion avoidance mechanism implemented in routers which works on the basis of active queue management. RED addresses the shortcomings of Tail Drop. A RED router signals incipient congestion to TCP by dropping packets probabilistically before the queue runs out of buffer space. This drop probability is dependent on a running average queue size to avoid any bias against bursty traffic. A RED router randomly drops arriving packets, with the result that the probability of dropping a packet belonging to a particular flow is approximately proportional to the flow's share of bandwidth. Thus, if the sender is using relatively more bandwidth it gets penalized by having more of its packets dropped. RED operates by maintaining two levels of thresholds minimum (minth) and maximum (maxth). It drops a packet probabilistically if and only if the average queue size lies between the minth and maxth thresholds. If the average queue size is above the maximum threshold, the arriving packet is always dropped. When the average queue size is between the minimum and the maximum threshold, each arriving packet is dropped with probability p_a , where p_a is a function of the average queue size. As the average queue length varies between minth and maxth, p_a increases linearly towards a configured maximum drop probability, maxp. Beyond maxth, the drop

probability is 100%. Dropping packets in this way ensures that when some subset of the source TCP packets get dropped and they invoke congestion avoidance algorithms that will ease the congestion at the gateway. Since the dropping is distributed across flows, the problem of global synchronization is avoided.

3. Explicit Congestion Notification

Explicit Congestion Notification is an extension proposed to RED which marks a packet instead of dropping it when the average queue size is between minth and maxth [7]. Since ECN marks packets before congestion actually occurs, this is useful for protocols like TCP that are sensitive to even a single packet loss. Upon receipt of a congestion marked packet, the TCP receiver informs the sender (in the subsequent ACK) about incipient congestion which will in turn trigger the congestion avoidance algorithm at the sender. ECN requires support from both the router as well as the end hosts, i.e. the end hosts TCP stack needs to be modified. Packets from flows that are not ECN capable will continue to be dropped by RED (as was the case before ECN).

3.1. Changes at the router

Router side support for ECN can be added by modifying current RED implementations. For packets from ECN capable hosts, the router marks the packets rather than dropping them (if the average queue size is between minth and maxth). It is necessary that the router identifies that a packet is ECN capable, and should only mark packets that are from ECN capable hosts. This uses two bits in the IP header. The ECN Capable Transport (ECT) bit is set by the sender end system if both the end systems are ECN capable (for a unicast transport, only if both end systems are ECN-capable). In TCP this is confirmed in the pre-negotiation during the connection setup phase (explained in Section 3.2). Packets encountering congestion are marked by the router using the Congestion Experienced (CE) (if the average queue size is between minth and maxth) on their way to the receiver end system (from the sender end system), with a probability proportional to the average queue size following the procedure used in RED (RFC2309) routers. Bits 10 and 11 in the IPV6 header are proposed respectively for the ECT and CE bits. Bits 6 and 7 of the IPV4 header DSCP field are also specified for experimental purposes for the ECT and CE bits respectively.

3.2. Changes at the TCP Host side

The proposal to add ECN to TCP specifies two new flags in the reserved field of the TCP header. Bit 9 in the reserved field of the TCP header is designated as the ECN-Echo (ECE) flag and Bit 8 is

designated as the Congestion Window Reduced (CWR) flag. These two bits are used both for the initializing phase in which the sender and the receiver negotiate the capability and the desire to use ECN, as well as for the subsequent actions to be taken in case there is congestion experienced in the network during the established state.

There are two main changes that need to be made to add ECN to TCP to an end system and one extension to a router running RED.

1. In the connection setup phase, the source and destination TCPs have to exchange information about their desire and/or capability to use ECN. This is done by setting both the ECN-Echo flag and the CWR flag in the SYN packet of the initial connection phase by the sender; on receipt of this SYN packet, the receiver will set the ECN-Echo flag in the SYN-ACK response. Once this agreement has been reached, the sender will thereon set the ECT bit in the IP header of data packets for that flow, to indicate to the network that it is capable and willing to participate in ECN. The ECT bit is set on all packets other than pure ACK's.

2. When a router has decided from its active queue management mechanism, to drop or mark a packet, it checks the IP-ECT bit in the packet header. It sets the CE bit in the IP header if the IP-ECT bit is set. When such a packet reaches the receiver, the receiver responds by setting the ECN-Echo flag (in the TCP header) in the next outgoing ACK for the flow. The receiver will continue to do this in subsequent ACKs until it receives from the sender an indication that it (the sender) has responded to the congestion notification.

3. Upon receipt of this ACK, the sender triggers its congestion avoidance algorithm by halving its congestion window, *cwnd*, and updating its congestion window threshold value *ssthresh*. Once it has taken these appropriate steps, the sender sets the CWR bit on the next data outgoing packet to tell the receiver that it has reacted to the (receiver's) notification of congestion. The receiver reacts to the CWR by halting the sending of the congestion notifications (ECE) to the sender if there is no new congestion in the network.

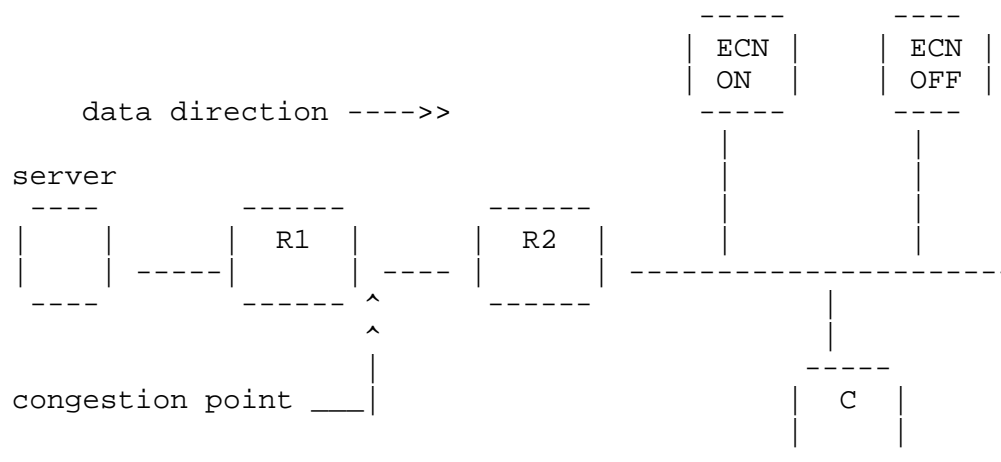
Note that the sender reaction to the indication of congestion in the network (when it receives an ACK packet that has the ECN-Echo flag set) is equivalent to the Fast Retransmit/Recovery algorithm (when there is a congestion loss) in NON-ECN-capable TCP i.e. the sender halves the congestion window *cwnd* and reduces the slow start threshold *ssthresh*. Fast Retransmit/Recovery is still available for ECN capable stacks for responding to three duplicate acknowledgments.

4. Experimental setup

For testing purposes we have added ECN to the Linux TCP/IP stack, kernels version 2.0.32, 2.2.5, 2.3.43 (there were also earlier revisions of 2.3 which were tested). The 2.0.32 implementation conforms to RFC 2481 [7] for the end systems only. We have also modified the code in the 2.1, 2.2 and 2.3 cases for the router portion as well as end system to conform to the RFC. An outdated version of the 2.0 code is available at [18]. Note Linux version 2.0.32 implements TCP Reno congestion control while kernels $\geq 2.2.0$ default to New Reno but will opt for a SACK/FACK combo when the remote end understands SACK. Our initial tests were carried out with the 2.0 kernel at the end system and 2.1 (pre 2.2) for the router part. The majority of the test results here apply to the 2.0 tests. We did repeat these tests on a different testbed (move from Pentium to Pentium-II class machines) with faster machines for the 2.2 and 2.3 kernels, so the comparisons on the 2.0 and 2.2/3 are not relative.

We have updated this memo release to reflect the tests against SACK and New Reno.

4.1. Testbed setup



The figure above shows our test setup.

All the physical links are 10Mbps ethernet. Using Class Based Queuing (CBQ) [22], packets from the data server are constricted to a 1.5Mbps pipe at the router R1. Data is always retrieved from the server towards the clients labelled, "ECN ON", "ECN OFF", and "C". Since the pipe from the server is 10Mbps, this creates congestion at the exit from the router towards the clients for competing flows. The machines labeled "ECN ON" and "ECN OFF" are running the same version

of Linux and have exactly the same hardware configuration. The server is always ECN capable (and can handle NON ECN flows as well using the standard congestion algorithms). The machine labeled "C" is used to create congestion in the network. Router R2 acts as a path-delay controller. With it we adjust the RTT the clients see. Router R1 has RED implemented in it and has capability for supporting ECN flows. The path-delay router is a PC running the Nistnet [16] package on a Linux platform. The latency of the link for the experiments was set to be 20 millisecs.

4.2. Validating the Implementation

We spent time validating that the implementation was conformant to the specification in RFC 2481. To do this, the popular tcpdump sniffer [24] was modified to show the packets being marked. We visually inspected tcpdump traces to validate the conformance to the RFC under a lot of different scenarios. We also modified tcptrace [25] in order to plot the marked packets for visualization and analysis.

Both tcpdump and tcptrace revealed that the implementation was conformant to the RFC.

4.3. Terminology used

This section presents background terminology used in the next few sections.

* Congesting flows: These are TCP flows that are started in the background so as to create congestion from R1 towards R2. We use the laptop labeled "C" to introduce congesting flows. Note that "C" as is the case with the other clients retrieves data from the server.

* Low, Moderate and High congestion: For the case of low congestion we start two congesting flows in the background, for moderate congestion we start five congesting flows and for the case of high congestion we start ten congesting flows in the background.

* Competing flows: These are the flows that we are interested in. They are either ECN TCP flows from/to "ECN ON" or NON ECN TCP flows from/to "ECN OFF".

* Maximum drop rate: This is the RED parameter that sets the maximum probability of a packet being marked at the router. This corresponds to maxp as explained in Section 2.1.

Our tests were repeated for varying levels of congestion with varying maximum drop rates. The results are presented in the subsequent sections.

* Low, Medium and High drop probability: We use the term low probability to mean a drop probability maxp of 0.02, medium probability for 0.2 and high probability for 0.5. We also experimented with drop probabilities of 0.05, 0.1 and 0.3.

* Goodput: We define goodput as the effective data rate as observed by the user, i.e., if we transmitted 4 data packets in which two of them were retransmitted packets, the efficiency is 50% and the resulting goodput is $2 \times \text{packet size} / \text{time taken to transmit}$.

* RED Region: When the router's average queue size is between minth and maxth we denote that we are operating in the RED region.

4.4. RED parameter selection

In our initial testing we noticed that as we increase the number of congesting flows the RED queue degenerates into a simple Tail Drop queue. i.e. the average queue exceeds the maximum threshold most of the times. Note that this phenomena has also been observed by [5] who proposes a dynamic solution to alleviate it by adjusting the packet dropping probability "maxp" based on the past history of the average queue size. Hence, it is necessary that in the course of our experiments the router operate in the RED region, i.e., we have to make sure that the average queue is maintained between minth and maxth. If this is not maintained, then the queue acts like a Tail Drop queue and the advantages of ECN diminish. Our goal is to validate ECN's benefits when used with RED at the router. To ensure that we were operating in the RED region we monitored the average queue size and the actual queue size in times of low, moderate and high congestion and fine-tuned the RED parameters such that the average queue zones around the RED region before running the experiment proper. Our results are, therefore, not influenced by operating in the wrong RED region.

5. The Experiments

We start by making sure that the background flows do not bias our results by computing the fairness index [12] in Section 5.1. We proceed to carry out the experiments for bulk transfer presenting the results and analysis in Section 5.2. In Section 5.3 the results for transactional transfers along with analysis is presented. More details on the experimental results can be found in [27].

5.1. Fairness

In the course of the experiments we wanted to make sure that our choice of the type of background flows does not bias the results that we collect. Hence we carried out some tests initially with both ECN and NON ECN flows as the background flows. We repeated the experiments for different drop probabilities and calculated the fairness index [12]. We also noticed (when there were equal number of ECN and NON ECN flows) that the number of packets dropped for the NON ECN flows was equal to the number of packets marked for the ECN flows, showing thereby that the RED algorithm was fair to both kind of flows.

Fairness index: The fairness index is a performance metric described in [12]. Jain [12] postulates that the network is a multi-user system, and derives a metric to see how fairly each user is treated. He defines fairness as a function of the variability of throughput across users. For a given set of user throughputs ($x_1, x_2 \dots x_n$), the fairness index to the set is defined as follows:

$$f(x_1, x_2, \dots, x_n) = \frac{\text{square}(\sum_{i=1..n} x_i)}{(n * \sum_{i=1..n} \text{square}(x_i))}$$

The fairness index always lies between 0 and 1. A value of 1 indicates that all flows got exactly the same throughput. Each of the tests was carried out 10 times to gain confidence in our results. To compute the fairness index we used FTP to generate traffic.

Experiment details: At time $t = 0$ we start 2 NON ECN FTP sessions in the background to create congestion. At time $t=20$ seconds we start two competing flows. We note the throughput of all the flows in the network and calculate the fairness index. The experiment was carried out for various maximum drop probabilities and for various congestion levels. The same procedure is repeated with the background flows as ECN. The fairness index was fairly constant in both the cases when the background flows were ECN and NON ECN indicating that there was no bias when the background flows were either ECN or NON ECN.

Max Drop Prob	Fairness With BG flows ECN	Fairness With BG flows NON ECN
0.02	0.996888	0.991946
0.05	0.995987	0.988286
0.1	0.985403	0.989726
0.2	0.979368	0.983342

With the observation that the nature of background flows does not alter the results, we proceed by using the background flows as NON ECN for the rest of the experiments.

5.2. Bulk transfers

The metric we chose for bulk transfer is end user throughput.

Experiment Details: All TCP flows used are RENO TCP. For the case of low congestion we start 2 FTP flows in the background at time 0. Then after about 20 seconds we start the competing flows, one data transfer to the ECN machine and the second to the NON ECN machine. The size of the file used is 20MB. For the case of moderate congestion we start 5 FTP flows in the background and for the case of high congestion we start 10 FTP flows in the background. We repeat the experiments for various maximum drop rates each repeated for a number of sets.

Observation and Analysis:

We make three key observations:

- 1) As the congestion level increases, the relative advantage for ECN increases but the absolute advantage decreases (expected, since there are more flows competing for the same link resource). ECN still does better than NON ECN even under high congestion. Inferring a sample from the collected results: at maximum drop probability of 0.1, for example, the relative advantage of ECN increases from 23% to 50% as the congestion level increases from low to high.
- 2) Maintaining congestion levels and varying the maximum drop probability (MDP) reveals that the relative advantage of ECN increases with increasing MDP. As an example, for the case of high congestion as we vary the drop probability from 0.02 to 0.5 the relative advantage of ECN increases from 10% to 60%.
- 3) There were hardly any retransmissions for ECN flows (except the occasional packet drop in a minority of the tests for the case of high congestion and low maximum drop probability).

We analyzed tcpdump traces for NON ECN with the help of tcptrace and observed that there were hardly any retransmits due to timeouts. (Retransmit due to timeouts are inferred by counting the number of 3 DUPACKS retransmit and subtracting them from the total recorded number of retransmits). This means that over a long period of time (as is the case of long bulk transfers), the data-driven loss recovery mechanism of the Fast Retransmit/Recovery algorithm is very effective. The algorithm for ECN on congestion notification from ECE

is the same as that for a Fast Retransmit for NON ECN. Since both are operating in the RED region, ECN barely gets any advantage over NON ECN from the signaling (packet drop vs. marking).

It is clear, however, from the results that ECN flows benefit in bulk transfers. We believe that the main advantage of ECN for bulk transfers is that less time is spent recovering (whereas NON ECN spends time retransmitting), and timeouts are avoided altogether. [23] has shown that even with RED deployed, TCP RENO could suffer from multiple packet drops within the same window of data, likely to lead to multiple congestion reactions or timeouts (these problems are alleviated by ECN). However, while TCP Reno has performance problems with multiple packets dropped in a window of data, New Reno and SACK have no such problems.

Thus, for scenarios with very high levels of congestion, the advantages of ECN for TCP Reno flows could be more dramatic than the advantages of ECN for NewReno or SACK flows. An important observation to make from our results is that we do not notice multiple drops within a single window of data. Thus, we would expect that our results are not heavily influenced by Reno's performance problems with multiple packets dropped from a window of data. We repeated these tests with ECN patched newer Linux kernels. As mentioned earlier these kernels would use a SACK/FAK combo with a fallback to New Reno. SACK can be selectively turned off (defaulting to New Reno). Our results indicate that ECN still improves performance for the bulk transfers. More results are available in the pdf version[27]. As in 1) above, maintaining a maximum drop probability of 0.1 and increasing the congestion level, it is observed that ECN-SACK improves performance from about 5% at low congestion to about 15% at high congestion. In the scenario where high congestion is maintained and the maximum drop probability is moved from 0.02 to 0.5, the relative advantage of ECN-SACK improves from 10% to 40%. Although these numbers are lower than the ones exhibited by Reno, they do reflect the improvement that ECN offers even in the presence of robust recovery mechanisms such as SACK.

5.3. Transactional transfers

We model transactional transfers by sending a small request and getting a response from a server before sending the next request. To generate transactional transfer traffic we use Netperf [17] with the CRR (Connect Request Response) option. As an example let us assume that we are retrieving a small file of say 5 - 20 KB, then in effect we send a small request to the server and the server responds by sending us the file. The transaction is complete when we receive the complete file. To gain confidence in our results we carry the simulation for about one hour. For each test there are a few thousand

of these requests and responses taking place. Although not exactly modeling HTTP 1.0 traffic, where several concurrent sessions are opened, Netperf-CRR is nevertheless a close approximation. Since Netperf-CRR waits for one connection to complete before opening the next one (0 think time), that single connection could be viewed as the slowest response in the set of the opened concurrent sessions (in HTTP). The transactional data sizes were selected based on [2] which indicates that the average web transaction was around 8 - 10 KB; The smaller (5KB) size was selected to guestimate the size of transactional processing that may become prevalent with policy management schemes in the diffserv [4] context. Using Netperf we are able to initiate these kind of transactional transfers for a variable length of time. The main metric of interest in this case is the transaction rate, which is recorded by Netperf.

* Define Transaction rate as: The number of requests and complete responses for a particular requested size that we are able to do per second. For example if our request is of 1KB and the response is 5KB then we define the transaction rate as the number of such complete transactions that we can accomplish per second.

Experiment Details: Similar to the case of bulk transfers we start the background FTP flows to introduce the congestion in the network at time 0. About 20 seconds later we start the transactional transfers and run each test for three minutes. We record the transactions per second that are complete. We repeat the test for about an hour and plot the various transactions per second, averaged out over the runs. The experiment is repeated for various maximum drop probabilities, file sizes and various levels of congestion.

Observation and Analysis

There are three key observations:

- 1) As congestion increases (with fixed drop probability) the relative advantage for ECN increases (again the absolute advantage does not increase since more flows are sharing the same bandwidth). For example, from the results, if we consider the 5KB transactional flow, as we increase the congestion from medium congestion (5 congesting flows) to high congestion (10 congesting flows) for a maximum drop probability of 0.1 the relative gain for ECN increases from 42% to 62%.
- 2) Maintaining the congestion level while adjusting the maximum drop probability indicates that the relative advantage for ECN flows increase. From the case of high congestion for the 5KB flow we

observe that the number of transactions per second increases from 0.8 to 2.2 which corresponds to an increase in relative gain for ECN of 20% to 140%.

3) As the transactional data size increases, ECN's advantage diminishes because the probability of recovering from a Fast Retransmit increases for NON ECN. ECN, therefore, has a huge advantage as the transactional data size gets smaller as is observed in the results. This can be explained by looking at TCP recovery mechanisms. NON ECN in the short flows depends, for recovery, on congestion signaling via receiving 3 duplicate ACKs, or worse by a retransmit timer expiration, whereas ECN depends mostly on the TCP-ECE flag. This is by design in our experimental setup. [3] shows that most of the TCP loss recovery in fact happens in timeouts for short flows. The effectiveness of the Fast Retransmit/Recovery algorithm is limited by the fact that there might not be enough data in the pipe to elicit 3 duplicate ACKs. TCP RENO needs at least 4 outstanding packets to recover from losses without going into a timeout. For 5KB (4 packets for MTU of 1500Bytes) a NON ECN flow will always have to wait for a retransmit timeout if any of its packets are lost. (This timeout could only have been avoided if the flow had used an initial window of four packets, and the first of the four packets was the packet dropped). We repeated these experiments with the kernels implementing SACK/FAK and New Reno algorithms. Our observation was that there was hardly any difference with what we saw with Reno. For example in the case of SACK-ECN enabling: maintaining the maximum drop probability to 0.1 and increasing the congestion level for the 5KB transaction we noticed that the relative gain for the ECN enabled flows increases from 47-80%. If we maintain the congestion level for the 5KB transactions and increase the maximum drop probabilities instead, we notice that SACKs performance increases from 15%-120%. It is fair to comment that the difference in the testbeds (different machines, same topology) might have contributed to the results; however, it is worth noting that the relative advantage of the SACK-ECN is obvious.

6. Conclusion

ECN enhancements improve on both bulk and transactional TCP traffic. The improvement is more obvious in short transactional type of flows (popularly referred to as mice).

* Because less retransmits happen with ECN, it means less traffic on the network. Although the relative amount of data retransmitted in our case is small, the effect could be higher when there are more contributing end systems. The absence of retransmits also implies an improvement in the goodput. This becomes very important for scenarios

where bandwidth is expensive such as in low bandwidth links. This implies also that ECN lends itself well to applications that require reliability but would prefer to avoid unnecessary retransmissions.

* The fact that ECN avoids timeouts by getting faster notification (as opposed to traditional packet dropping inference from 3 duplicate ACKs or, even worse, timeouts) implies less time is spent during error recovery - this also improves goodput.

* ECN could be used to help in service differentiation where the end user is able to "probe" for their target rate faster. Assured forwarding [1] in the diffserv working group at the IETF proposes using RED with varying drop probabilities as a service differentiation mechanism. It is possible that multiple packets within a single window in TCP RENO could be dropped even in the presence of RED, likely leading into timeouts [23]. ECN end systems ignore multiple notifications, which help in countering this scenario resulting in improved goodput. The ECN end system also ends up probing the network faster (to reach an optimal bandwidth). [23] also notes that RENO is the most widely deployed TCP implementation today.

It is clear that the advent of policy management schemes introduces new requirements for transactional type of applications, which constitute a very short query and a response in the order of a few packets. ECN provides advantages to transactional traffic as we have shown in the experiments.

7. Acknowledgements

We would like to thank Alan Chapman, Ioannis Lambadaris, Thomas Kunz, Biswajit Nandy, Nabil Seddigh, Sally Floyd, and Rupinder Makkar for their helpful feedback and valuable suggestions.

8. Security Considerations

Security considerations are as discussed in section 9 of RFC 2481.

9. References

- [1] Heinanen, J., Finland, T., Baker, F., Weiss, W. and J. Wroclawski, "Assured Forwarding PHB Group", RFC 2597, June 1999.
- [2] B.A. Mat. "An empirical model of HTTP network traffic." In proceedings INFOCOMM'97.

- [3] Balakrishnan H., Padmanabhan V., Seshan S., Stemn M. and Randy H. Katz, "TCP Behavior of a busy Internet Server: Analysis and Improvements", Proceedings of IEEE Infocom, San Francisco, CA, USA, March '98
<http://nms.lcs.mit.edu/~hari/papers/infocom98.ps.gz>
- [4] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z. and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, December 1998.
- [5] W. Feng, D. Kandlur, D. Saha, K. Shin, "Techniques for Eliminating Packet Loss in Congested TCP/IP Networks", U. Michigan CSE-TR-349-97, November 1997.
- [6] S. Floyd. "TCP and Explicit Congestion Notification." ACM Computer Communications Review, 24, October 1994.
- [7] Ramakrishnan, K. and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP", RFC 2481, January 1999.
- [8] Kevin Fall, Sally Floyd, "Comparisons of Tahoe, RENO and Sack TCP", Computer Communications Review, V. 26 N. 3, July 1996, pp. 5-21
- [9] S. Floyd and V. Jacobson. "Random Early Detection Gateways for Congestion Avoidance". IEEE/ACM Transactions on Networking, 3(1), August 1993.
- [10] E. Hashem. "Analysis of random drop for gateway congestion control." Rep. Lcs tr-465, Lav. Fot Comput. Sci., M.I.T., 1989.
- [11] V. Jacobson. "Congestion Avoidance and Control." In Proceedings of SIGCOMM '88, Stanford, CA, August 1988.
- [12] Raj Jain, "The art of computer systems performance analysis", John Wiley and sons QA76.9.E94J32, 1991.
- [13] T. V. Lakshman, Arnie Neidhardt, Teunis Ott, "The Drop From Front Strategy in TCP Over ATM and Its Interworking with Other Control Features", Infocom 96, MA28.1.
- [14] P. Mishra and H. Kanakia. "A hop by hop rate based congestion control scheme." Proc. SIGCOMM '92, pp. 112-123, August 1992.
- [15] Floyd, S. and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 2582, April 1999.

- [16] The NIST Network Emulation Tool
<http://www.antd.nist.gov/itg/nistnet/>
- [17] The network performance tool
<http://www.netperf.org/netperf/NetperfPage.html>
- [18] <ftp://ftp.ee.lbl.gov/ECN/ECN-package.tgz>
- [19] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J. and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.
- [20] K. K. Ramakrishnan and R. Jain. "A Binary feedback scheme for congestion avoidance in computer networks." ACM Trans. Comput. Syst., 8(2):158-181, 1990.
- [21] Mathis, M., Mahdavi, J., Floyd, S. and A. Romanow, "TCP Selective Acknowledgement Options", RFC 2018, October 1996.
- [22] S. Floyd and V. Jacobson, "Link sharing and Resource Management Models for packet Networks", IEEE/ACM Transactions on Networking, Vol. 3 No.4, August 1995.
- [23] Prasad Bagal, Shivkumar Kalyanaraman, Bob Packer, "Comparative study of RED, ECN and TCP Rate Control".
<http://www.packeteer.com/technology/Pdf/packeteer-final.pdf>
- [24] tcpdump, the protocol packet capture & dumper program.
<ftp://ftp.ee.lbl.gov/tcpdump.tar.Z>
- [25] TCP dump file analysis tool:
<http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html>
- [26] Thompson K., Miller, G.J., Wilder R., "Wide-Area Internet Traffic Patterns and Characteristics". IEEE Networks Magazine, November/December 1997.
- [27] <http://www7.nortel.com:8080/CTL/ecnperf.pdf>

10. Authors' Addresses

Jamal Hadi Salim
Nortel Networks
3500 Carling Ave
Ottawa, ON, K2H 8E9
Canada

EMail: hadi@nortelnetworks.com

Uvaiz Ahmed
Dept. of Systems and Computer Engineering
Carleton University
Ottawa
Canada

EMail: ahmed@sce.carleton.ca

11. Full Copyright Statement

Copyright (C) The Internet Society (2000). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

