On the Prevalence and Evaluation of Recent TCP Enhancements

Sourabh Ladha Paul D. Amer Armando Caro Jr. Janardhan R. Iyengar
Protocol Engineering Lab,
Computer and Information Sciences
University of Delaware
{ladha, amer, acaro, iyengar}@cis.udel.edu

***Abstract* - In recent years several enhancements to TCP congestion control and loss recovery mechanisms have been proposed and accepted as Internet standards. While each proposal has been accompanied with related research, a number of questions remain to be answered both in the research and the implementer community: (i) What is the current deployment status of these TCP enhancements in the Internet, (ii) What is the effect of TCP enhancements on web based transfers, and (iii) How do bulk data transfers benefit from the cumulative addition of these TCP enhancements. In this paper, we attempt to answer these questions. We consider five TCP enhancements: (1) Selective Acknowledgements (SACK) and the SACK-based loss recovery algorithm, (2) Increasing Initial Congestion Window, (3) Limited Transmit, (4) Appropriate Byte Counting, and (5) Early Retransmit. We present results from active measurements performed on web servers on the state of deployment of these enhancements. Our results show that while several web servers support TCP enhancements, the majority still use previous standards for congestion control and loss recovery. Using simulation, we quantify the cumulative effect of these TCP enhancements on web based and bulk data transfers. We hope that such an evaluation provides a clearer view of the applicability of these enhancements, and further motivation for their implementation.**

I. INTRODUCTION

End-to-end congestion control and loss recovery has interested the research community for nearly two decades, and still remains one of the most prominent areas of networking research. TCP, the Internet's prevalent transport protocol [MC00], uses congestion control and loss recovery mechanisms first defined in [JK88] and standardized in [APS99]. Recently, five modifications to TCP congestion control and loss recovery mechanisms: (1) Selective Acknowledgments (SACK) and the SACK-based loss recovery algorithm [MMF+96, BAF+03], (2) Increasing Initial Congestion Window [AFP+02], (3) Limited Transmit [ABF01], (4) Appropriate Byte Counting [All03], and (5) Early Retransmit [AAA+03], have been proposed and accepted as Internet standards. The focus of these modifications has been to reduce the retransmission timeouts in TCP or to perform finer loss recovery. A cumulative assessment of these mechanisms has been lacking and requires attention due to the following reasons. First, implementers and vendors need to know the performance incentives for implementing these enhancements in their TCP stacks and in the context of their requirements (i.e., for the category of transfers (mice or elephants) they are interested in). Second, researchers who base their experiments on TCP need to know if their benchmarks reflect the current TCP that is deployed in the Internet. In this paper we attempt to answer these questions.

Several methodologies to measure the support of different TCP mechanisms at a remote host have been developed over the years. One such methodology is the TCP Behavior Inference Tool (TBIT) [TBIT, PF01], which measures the existence of TCP options and features in remote web servers. TBIT establishes a user level TCP connection to a remote web server and generates requests for a base web page. Using the BSD packet filter device [MJ92], TBIT captures the incoming packets at the user level and prevents them from reaching the kernel. Using special sequences of packets and by monitoring the incoming packets, TBIT identifies the TCP protocol features of the remote web server. The TBIT packets generated cannot be distinguished from legitimate user traffic, thus providing an advantage over tools such as NMAP [NMAP] that use special packets to perform TCP fingerprinting. Thus, TBIT packets have a minimal chance of being blocked by a server firewall. We developed and implemented TBIT extensions for Limited Transmit, Appropriate Byte Counting, and Early Retransmit and used the already implemented tests for determining Initial Congestion Window and SACK support. We report on the prevalence of these TCP enhancements in a small web server space in the Internet. We hope that these results will help researchers update the benchmarks of their experiments on the TCP variant prevalent in the Internet [AF99].

Each TCP enhancement discussed in this paper proposes to improve congestion control and loss recovery. For example, RFC3390 [AFP+02] for Increasing TCP's Initial Window allows the limit on the initial window of a new TCP connection to be increased up to 4K bytes, or roughly three to four packets. Such an increase helps short transfers and has minimal effect on bulk data transfers. While each of these standards has been accompanied with related research [All98, Bal98, PN98, FF96], the cumulative assessment of TCP with all the enhancements is lacking. In this paper we present an evaluation of current TCP by benchmarking TCP NewReno and comparing it with the cumulative addition of TCP enhancements. We present this evaluation for web based and bulk data transfers with different transfer sizes. We believe that such an evaluation will provide useful information to implementers and vendors for considering these enhancements.

The remainder of this paper is organized as follows. Section II presents a brief description of the five TCP enhancements under study. Section III describes the TBIT extensions and the results obtained from the tests applied to web servers in the Internet. Section IV describes the simulation methodology and evaluation results. Section V concludes the paper with a discussion on ongoing and future work.

*Prepared through collaborative participation in the Communication and Network Consortium sponsored by the U.S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.
* Research supported, in part, by the University Research Program, Cisco Systems, Inc.

II. TCP ENHANCEMENTS

TCP SACK as defined in [MMF+96] extends TCP to include the new SACK option. TCP SACK receivers can report multiple out-of-order received packets via this SACK option. TCP SACK senders retransmit only those packets that have not been acknowledged via the SACK option. In TCP without SACK, a sender can determine only a single packet loss per round trip time (RTT). TCP SACK allows a sender to recover from multiple losses in a single RTT. Measurement studies showed that while several servers in the Internet were advertising the SACK option, the SACK information was not being used to perform finer loss recovery [PF01]. This discovery pressed the need for laying out a standard based on [FF96] that presents a conservative loss recovery mechanism for TCP SACK [BAF+03].

TCP congestion control as defined in [APS99], restricted TCP implementations from increasing their initial congestion window (cwnd) beyond twice the Maximum Segment Size (MSS). Several scenarios demanded an increase in the value of initial cwnd: (i) short web based transfers where the web page transfer time is on order of RTTs, (ii) Long delay TCP connections, as over satellite channels, where RTTs are in order of seconds, and hence lead to slower evolution of the cwnd. A new standard specified in [AFP+02] increases the limit of 2*MSS for the initial cwnd up to 4380 bytes or roughly three to four packets. The minimum initial cwnd that can be used by a TCP connection was specified in [AFP+02] to be two packets, thus reducing the chances of expiring the delayed Ack timer of the TCP receiver for the first window of packets.

A TCP receiver on receiving out-of-order packets sends duplicate acknowledgments (Acks) cumulatively acknowledging the last in-order segment received. A TCP sender infers loss of data either if it receives a threshold number of duplicate Acks from the receiver, or if it does not receive any feedback from the receiver within a retransmission time out (RTO) interval. The duplicate Ack threshold of inferring loss is currently specified as three [APS99]. A TCP sender is not allowed to send new data on the receipt of duplicate Acks. This leads to scenarios where a sender does not have enough data outstanding that could generate three duplicate Acks if a loss occurs, limiting the sender to infer the loss only after RTO amount of time. A new standard, Limited Transmit [ABF01], allows TCP senders to send new data on the receipt of first two duplicate Acks. Limited transmit increases the probability of a sender to receive three duplicate Acks when a loss does occur, and decreases the chances of an RTO.

TCP implementations typically maintain the congestion window for a connection in packets as opposed to bytes. On receipt of an Ack that acknowledges new data, TCP senders increase the cwnd by one packet in slow start, and 1/cwnd packets in congestion avoidance. Misbehaving or greedy receivers can generate multiple "split" Acks for a single data packet, thus making the sender increase its cwnd by several packets [SCW+99]. This aggressive behavior is inappropriate as it may result in unfair sharing of the network resources between behaving and misbehaving flows. Appropriate Byte Counting (ABC) [All03], recently standardized, calls for TCP senders to maintain the congestion window in bytes rather than packets and base the increase of cwnd on the number of bytes being acknowledged. While protecting against misbehaving "split-acking" receivers, ABC also improves the evolution of cwnd in the case of Ack loss. ABC defines a limit L that is the maximum amount by which a single Ack can increase the sender's cwnd. The standard recommends the conservative value of L=1*MSS, and allows an experimental value of L=2*MSS.

While Limited Transmit helps to reduce the chances of a timeout when the number of outstanding segments is less than enough to generate three duplicate Acks, it requires a TCP sender to have new data to send on the receipt of first two duplicate Acks. Application limited flows that generate small bursts of traffic may not be able to take advantage of Limited Transmit and may have to rely on an RTO to detect packet loss. A recent proposal called Early Retransmit [AAA+03] allows application limited TCP senders to retransmit data on the receipt of less than three duplicate Acks, in the hope of avoiding an expensive RTO. Early Retransmit is currently an Internet draft within the IETF [AAA+03] and is in the last phases of being standardized.

III. ACTIVE MEASUREMENTS

We developed TBIT tests to measure the prevalence of TCP enhancements among web servers in the Internet. We used the Initial Congestion Window and SACK tests that were already developed in the base version of TBIT for our measurements. In the following discussion we report on the design of the tests and the results obtained. We assume that the reader is familiar with the basic design and architecture of TBIT as described in [TBIT, PF01].

A. Methodology

1. Limited Transmit

The TBIT test to determine if a remote server supports Limited Transmit requires that TBIT be aware of the Initial Congestion Window (ICW) being used by the remote server. Once the ICW is determined the following test can be used to probe a web server for Limited Transmit support.

- TBIT establishes a TCP connection with the remote server.
- TBIT sends the base web page request to the remote server.
- The remote server starts sending the base web page to the TBIT client.
- TBIT sends an Ack for the 1st packet.
- TBIT drops the 2nd packet.
- TBIT sends a duplicate Ack (indicating it is waiting for packet 2) in response to the 3rd packet.
- TBIT does not acknowledge any further packets.
- TBIT monitors the incoming packets incrementing the "highest packet number" for every new data packet received until the 1st retransmission.
- A remote server of a given ICW supports Limited Transmit when the highest packet number received by TBIT is:
  (i) ICW = 1, highest packet = 4
  (ii) ICW = 2, highest packet = 5
  …
  (n) ICW = n, highest packet = n+3
- If the highest packet received from a server of ICW=' n' is less than n+3, Limited Transmit is not supported by the remote server;
- If the highest packet received from a server of ICW=' n' is greater than n+3, TBIT test exits without conclusion;

2. Appropriate Byte Counting (ABC)

We tested the prevalence of ABC in TCP web servers using two tests. The first test determines if a remote server supports ABC. The second test determines the value of L being used by the servers that support ABC. Both tests require that TBIT be aware of the ICW

being used by the remote server. The first TBIT test to determine if a remote server supports ABC is:

- TBIT establishes a TCP connection with the remote server.
- TBIT sends the base web page request to the remote server.
- The remote server starts sending the base web page.
- TBIT sends two split Acks for the 1st packet, each of them acknowledging half of the 1st packet.
- TBIT does not acknowledge any more packets.
- TBIT monitors the incoming packets incrementing the "highest packet number" for every new data packet received until the 1st retransmission.
- If the retransmission contains a sequence number less than that of the 2nd packet, TBIT exits without conclusion.
- A remote server of a given ICW does not support ABC when the highest packet number received by TBIT is:
  (i) ICW = 1, highest packet > 3
  (ii) ICW = 2, highest packet > 4
  …
  (n) ICW = n, highest packet > n+2
- If for an ICW of 'n' packets, the highest packet number received is less than 'n+3', the remote server supports ABC.

The second TBIT test for ABC is used to measure the value of L being used by the servers supporting ABC. The test is as follows.

- TBIT establishes a TCP connection with the remote server
- TBIT sends the base web page request to the remote server.
- The remote server starts sending the base web page.
- TBIT sends an Ack for the 1st packet.
- TBIT sends a delayed Ack for the 2nd and 3rd packet.
- TBIT does not acknowledge any more packets.
- TBIT monitors the incoming packets incrementing the "highest packet number" for every new data packet received till the 1st retransmission.
- If the retransmission contains a sequence number less than that of the 4th packet, TBIT exits without conclusion.
- A remote server of a given ICW uses a value of L for Appropriate Byte Counting (ABC) when the highest packet number received by TBIT is:
  (i) ICW = 1, highest packet = 6, L=1; highest packet = 7, L=2
  (ii) ICW = 2, highest packet = 7, L=1; highest packet = 8, L=2
  …
  (n) ICW=n, highest packet=n+5, L=1; highest packet=n+6, L=2
- If the highest packet received for ICW=' n' is greater than n+6, TBIT test exits without conclusion;
- If the highest packet received for ICW=' n' is less than n+5, TBIT test exits without conclusion;

3. Early Retransmit

The Early Retransmit mechanism gets invoked in scenarios where a TCP sender does not have any new data to send and the amount of outstanding data is less than 4*MSS. To generate this scenario, the TBIT test required a special byte-range HTTP request [FGM+99] to be generated by the TBIT client. The byte range request will ensure that the server sends less than 4*MSS packets, to cause a scenario where the support for Early Retransmit can be tested. The description of the test is as follows.

- TBIT establishes a TCP connection with the remote server
- TBIT sends the base page byte range request to the remote server, requesting 2*MSS bytes (This will generate 3 packets from the remote server: 2*MSS bytes of data + HTTP header).
- The remote server starts sending the base web page.
- TBIT sends an Ack for the 1st packet.
- TBIT drops the 2nd packet.
- TBIT sends a duplicate Ack (indicating it is waiting for packet 2) in response to the 3rd packet.
- In sequence TBIT now sends an Ack for all the packets.
- TBIT monitors the incoming packets.
- If TBIT receives a retransmission for the 2nd packet, the remote server supports Early Retransmit;
  else, the remote server does not support Early Retransmit;

B. Results

We performed the TBIT tests described above on a list of top 500 global web sites based on the rankings by Alexa's website [ALEXA]. Clearly this sample is not statistically significant[1]. For every server, we ran each TBIT test five times. We considered the test on a server valid if and only if four or more of these five tests returned the same result. The remaining servers were eliminated from the results for that particular test. Possible sources of error included packet reordering, unexpected drop, no response from the server, besides others [PF01]. The MSS used for the tests was 128 bytes with an exception for the Early Retransmit test that used an MSS of 1000 bytes. The results of the test are shown in Table 1.

We first ran the Initial Congestion Window test on the list of web servers. We found that 62 of the 423 web servers that returned results still used an initial window of one packet. This small number is surprising as TCP congestion control standardized the use of initial window of 2 packets several years back [APS99]. Of the remaining 361 web servers, 311 used an initial window of two packets. The use of initial congestion windows higher than two was found to be minimal. While 25 web servers used an initial window of three packets, only 17 web servers used initial window of four

TABLE I
TBIT TEST RESULTS

Category of TBIT test/ TCP feature	Number of web servers supporting the feature
Initial Window (= 1)	62
Initial Window (= 2)	311
Initial Window (= 3)	25
Initial Window (= 4)	17
Initial Window (> 4)	8
SACK Advertised	344
SACK Information Used	90
Limited Transmit Supported	99
Appropriate Byte Counting (ABC)	100
ABC with L = 1	80
ABC with L = 2	0
Early Retransmit	0

[1] We are currently running the TBIT tests for TCP enhancements on a list of 27000 web servers. The results from these tests will be included in the final paper, if accepted.

packets. The maximum initial window allowed by [AFP+02], for a TCP connection with an MSS of 128 bytes is four packets. Few web servers used an initial window of more than 4 packets. Interestingly, the Center of Information Technology's website at the National Institute of Health (http://www.cit.nih.gov) used the highest initial window of 23 packets! We are currently using fingerprinting tools to determine the operating systems being run by the web servers.

Surprisingly, a large numbers of servers advertised the SACK option but less than one third of these servers utilized the SACK information to perform sender side loss recovery. We found that 344 out of the 486 web servers that returned results advertised the SACK option. The remaining 142 web servers did not advertise the SACK option and hence were not included in the SACK utilization test. We tested the 344 web servers that advertised SACK for sender side SACK behavior. Only 90 out of the 344 web servers utilized the SACK information to perform sender side loss recovery.

The Limited Transmit test required that the initial window of the remote server be known. From the Initial Window test, we knew the initial window being used by 423 web servers. 224 out of the 423 web servers returned results for the Limited Transmit test. We found that 99 of the 224 web servers that returned results supported Limited Transmit, while 125 web servers did not support Limited Transmit.

The ABC tests required that the initial window of the remote server be known. Thus the test to determine if a remote server supports ABC was run on a list of 423 web servers. Out of the 200 servers that returned results, 100 supported ABC, while the remaining 100 did not support ABC. It is surprising that even though the threat of a misbehaving receiver is large, 50% of the servers do not support ABC. A misbehaving receiver (i.e., client) can send out multiple split Acks for a single data packet and make a non-ABC sender (i.e. server) open up its congestion window unfairly in comparison to conforming flows. Our next test involved testing the value of the limit L (as defined in [All03]) being used by servers that support ABC. We found that out of the 80 servers that returned results, all of them used L = 1. None of the servers used L = 2. This result is in spirit of the recommendation given in [All03].

The Early Retransmit test was run on the entire list of 500 web servers. The Early Retransmit test required that the server supports byte-range request specified in HTTP/1.1 [FGM+99]. Out of the 500 web servers only 80 servers returned results, since most of the servers did not support the byte-range request. None of the 80 web servers supported Early Retransmit!

IV. Simulation Results And Analysis

A. Methodology

To assess the cumulative effects of the five TCP protocol changes we performed simulations using the Network Simulator (ns2) [NS]. The topology used for the simulations is illustrated in Fig. 1. All the links used in the topology are full duplex. A single TCP source (***TCP Sender*** in the topology) is connected to the drop-tail router ***R1***. The link connecting the TCP source to the router is of capacity 5Mbps and has a one-way propagation delay of 1ms. A single TCP sink (***TCP Receiver*** in the topology) is connected to the drop-tail router ***R2***. The link connecting the TCP sink to the router is of capacity 5Mbps and has a one-way propagation delay of 1ms. To resemble the observed nature of traffic on data networks [LTW+93], we use self-similar cross-traffic. To generate self-similar cross-traffic, we use four cross-traffic nodes, each having eight Pareto ON-OFF traffic generators connected to routers ***R1*** and ***R2.*** Each Pareto source has an average ON time of 1ms and OFF time of 9ms. Each cross-traffic node is connected to a router (***R1*** or ***R2***) via a 5Mbps link with a propagation delay randomized to be between 1ms and 5ms. The Pareto sources connected to the router ***R1*** generate the forward path cross-traffic for the TCP data flow, and the Pareto sources connected to the router ***R2*** generate the reverse path cross-traffic for the TCP Ack flow. The cross-traffic packet sizes are chosen to resemble the distribution found in the Internet [CAIDA]: 50% are 44B, 25% are 576B, and 25% are 1500B. Thus the average packet size for the cross-traffic is 541B. A link of capacity 1Mbps with a one-way propagation delay of 35ms (approximate propagation delay between US coast to coast) forms the core link connecting router ***R1*** and ***R2***. The buffer capacity at routers ***R1*** and ***R2*** for the core link was set to twice the bandwidth-delay product (BDP) of the core link. Using the 2*BDP product and the average packet size of 541B, the buffer size at each of the routers ***R1*** and ***R2*** was set to 16 packets.

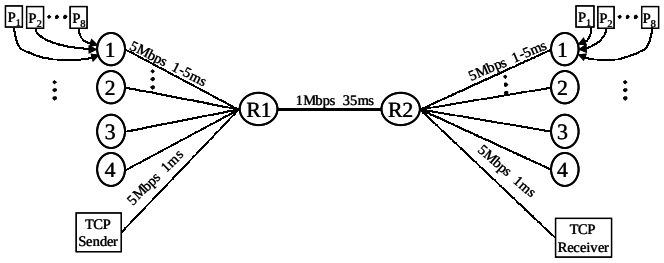


Fig. 1. Network topology for simulations

The Maximum Transmission Unit (MTU) for all links was set to the standard Ethernet MTU of 1500B. The TCP sender used an MSS of 1460B. The TCP receiver used delayed Acks with a delayed Ack timer of 200ms. For all simulations described in this paper, the cross-traffic was allowed to run for 10s before the TCP sender began sending data.

The parameters varied in the topology described in Fig. 1 were the aggregate rate of cross-traffic, the TCP variants, and the size of the file being transferred. The aggregate cross-traffic ranged from no cross-traffic to the bottleneck bandwidth of 1Mbps. The aggregate cross-traffic was found to be directly proportional to the number of packet losses generated. The packet losses in the simulations were only due to network congestion. The TCP variants were chosen using an incremental combination of the TCP enhancements considered in this paper. We benchmarked TCP NewReno as the TCP variant to base our comparisons. TCP NewReno had recently been observed to be the most popular TCP variant in the Internet [PF01]. Using TCP SACK as the starting point of the TCP enhancements in study, we cumulatively added each TCP enhancement in the order of their standardization in the IETF. The TCP SACK and TCP NewReno variants used an initial window of one packet. The TCP variants used in the simulations are as follows. The acronym in parentheses is used to refer to the respective variant in the rest of this paper.

(a) TCP NewReno {NewReno}
(b) TCP SACK {SACK}
(c) Increased Initial Window with (b) {SACK-IW}
(d) Limited Transmit with (c) {SACK-IW-LT}
(e) Appropriate Byte Counting (L=1) with (d) {SACK-IW-LT-ABC1}
(f) Early Retransmit with (e) {SACK-IW-LT-ABC1-ER}

B. Web Transfers

To assess the effect of TCP enhancements on web transfers, we show the results for transfer of a single 20K and a single 100K file. During the TBIT tests we observed that the base web page size used in the Internet has significant variance. A plain text web page could be as low as 10K in size while a web page with several objects (pictures, thumbnails, audio, etc) could go up to 500K in size. We thus chose a sample of 20K and 100K as representatives for web transfers. We varied the aggregate cross-traffic arrival rate to simulate packet drops for the TCP flow.

Fig. 2 shows the transfer time of a 20K file, as a function of the aggregate cross-traffic arrival rate, using the different TCP variants. Fig. 3(a) plots the number of packet drops seen by a TCP flow for different levels of cross-traffic. Each point in the graphs represents an average of multiple runs. The results for cross-traffic arrival rates below 800Kbps have not been plotted in Fig. 1 as the TCP flow did not experience any loss below 800Kbps, leading to nearly same transfer times for all the variants. At higher levels of cross-traffic NewReno slightly outperforms SACK. SACK-IW outperforms both SACK and NewReno and is the main contributor for better performance of the cumulative enhancements. This is because short transfers are on the order of few RTTs and saving one RTT by using larger a initial window significantly reduces transfer times. SACK-IW and all the later cumulative variants use an initial window of 3 packets as specified in [AFP+02], whereas NewReno and SACK use an initial window of 1 packet. SACK-IW-LT uses Limited Transmit, that allows to send new data on duplicate Acks. It can be seen from Fig. 1 that SACK-IW-LT outperforms SACK and NewReno for all levels of cross-traffic and performs almost the same or better than SACK-IW. The effect of ABC is minimal, especially for lower loss rates, as short transfers typically remain in slow start, while ABC with an L=1 is helpful for transfers that spend majority of their time in the congestion avoidance phase of TCP congestion control. In the next section of bulk data transfers, we notice the effect of ABC more prominently and explain the rationale behind it. In Fig. 2, the effect of Early Retransmit (ER) is also minimal as ER gets invoked only in the special case at the end of the file transfers. The effect of ER will be more predominant for transfers less than 4 packets (5K file transfers) where ER may be able to avoid expensive timeouts in the event of packet loss.

Fig. 3(b) shows the CDF of the number of times Limited Transmit (LT) and ER get invoked vs. the fraction of the total number of runs. The total number of runs, on which the fraction in

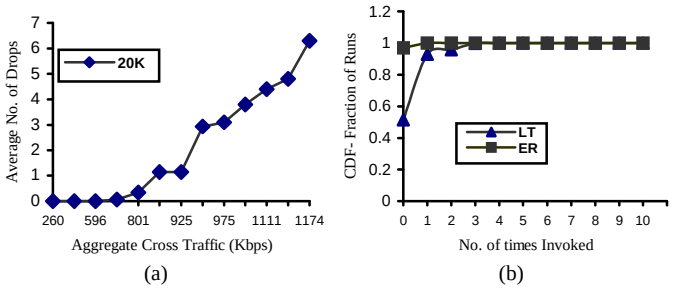


Fig. 3. (a) Aggregate Cross-Traffic (Kbps) vs. Average number of drops for a 20K transfer
(b) CDF of the Number of times Limited Transmit or Early Retransmit get invoked vs. Fraction of Runs for a 20K transfer.

Fig. 3(b) is calculated, is the collection of all LT and ER enabled runs for the LT and ER curve, respectively. As expected, ER gets invoked only once, and for only 3% of the runs. However, it is important to note that although ER occurs rather rarely, it does make its case as the 3% of the total runs where ER fast retransmits the missing packet, may have suffered a timeout without ER. For small file transfers, a single timeout could significantly increase the transfer time. For 51% of the runs, LT does not get invoked. This is because for cross-traffic levels below 800Kbps no losses were seen by the TCP flow. For the remaining 49% of the runs, LT gets invoked "at least" once. Each time LT gets invoked, a new data segment can be transmitted, thus improving (i.e. reducing) the total transfer time as seen in Fig. 1.

Fig. 4 shows the transfer time for a 100K file vs. the aggregate cross-traffic arrival rate. SACK outperforms NewReno slightly for cross-traffic rates beyond 800Kbps. The increase of initial cwnd has no significant effect on transfer time. LT and ABC reduce the transfer time by over 10 seconds for aggregate cross-traffic arrival rates of 1Mbps. While LT allows for sending new data on duplicate Acks yielding in increased goodput, ABC allows better evolution of cwnd in congestion avoidance. Without ABC, a TCP sender will increase its cwnd by one MSS in every two RTTs (due to delayed Acks), when in congestion avoidance. An ABC sender will be able to increase its cwnd by one MSS once per RTT as the cwnd is increased based on the number of bytes acknowledged rather than number of Acks received. The reverse path cross-traffic causes loss of Acks similar to the loss of data in the forward path. Loss of Acks in congestion avoidance may not allow cwnd to be increased once per two RTTs for a TCP sender without ABC. On the other hand a

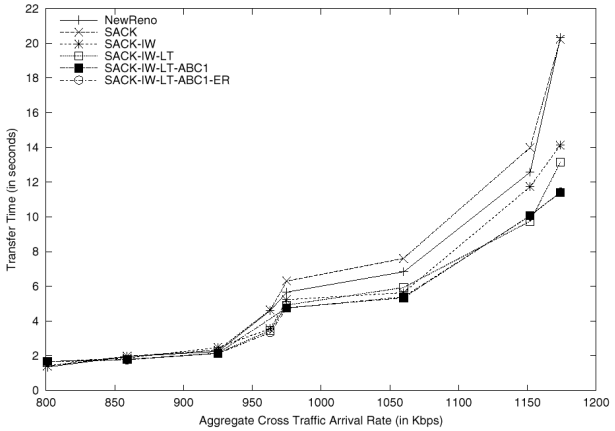


Fig. 2. Transfer Time vs. Aggregate Cross-Traffic for a 20K transfer

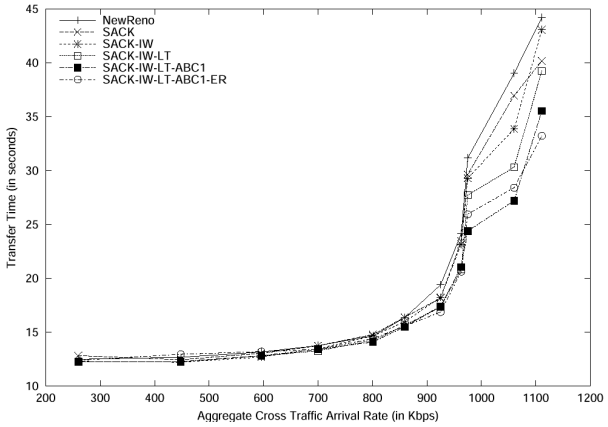
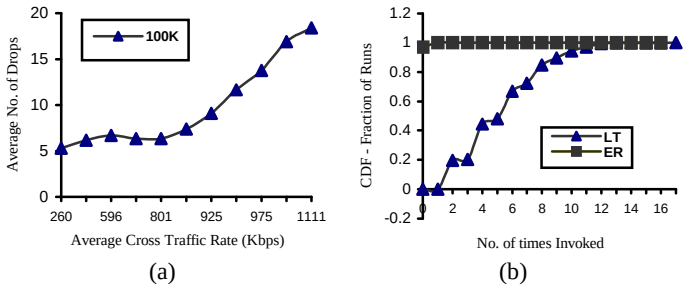


Fig. 4. Transfer Time vs. Aggregate Cross-Traffic for a 100K transfer



(a) (b)

Fig. 5. (a) Aggregate Cross-Traffic (Kbps) vs. Average number of drops for a 100K transfer
(b) CDF of the Number of times Limited Transmit or Early Retransmit get invoked vs. Fraction of Runs for a 100K transfer.

TCP sender with ABC is not affected by the loss of Acks in congestion avoidance as long as an Ack cumulatively acknowledging previous packets arrives. Since the 100K flow operates in congestion avoidance for most of its transfer, ABC causes better evolution of cwnd reducing the transfer time.

Fig. 5(a) shows the average number of data packet drops as a function of the cross-traffic arrival rate. Starting from 800Kbps of cross-traffic, the losses start increasing steadily, matched by the increased transfer time as seen in Figure 4. As the file size to be transferred increased from 20K to 100K, a greater number of LT events should be generated. This can be seen in Figure 5(b), where all sample runs generated at least two LT events, and approximately 20% of the runs generated more than eight LT events. ER gets invoked only once, and for only 2.5% of the runs.

B. Bulk Transfers

While web transfers present their own challenges, bulk transfers are also an important metric for evaluating the performance of TCP variants. We performed a bulk transfer of 10M to assess the throughput achieved by the TCP variants. The aggregate cross-traffic was varied to simulate packet drops.

Fig. 6 shows the transfer time taken by the TCP variants for a 10M transfer as a function of the loss rate as seen by the TCP flow. The loss rate is represented as a fraction of the total number of packets dropped out of the total number of "new" data packets sent. Fig. 6 shows that NewReno and SACK perform the same for loss rates below 9%, but as the loss rates increase further, SACK outperforms NewReno. While the loss rates beyond 10% may seem excessive for the Internet, the plot shown in Fig. 1 illustrates the pattern of performance of the TCP variants for bulk transfers. The SACK-IW curve lies exactly over the SACK curve thus showing that a higher initial cwnd makes little difference for bulk transfers. SACK-IW-LT yields in better transfer times for higher loss rates as well as for loss rates below 10%. Thus bulk transfers clearly gain by the Limited Transmit feature. Bulk transfers spend most of the connection lifetime in congestion avoidance. As described before ABC allows for better evolution of cwnd in congestion avoidance for receivers that generate delayed Acks and in situations of Ack loss. A TCP sender that supports ABC increases its cwnd by one MSS once per RTT, as opposed to once per two RTTs that a TCP sender without ABC would increase. Loss of Acks leads to yet slower cwnd evolution for a TCP sender without ABC. The performance improvement with ABC is evident in Fig. 6 where SACK-IW-LT-ABC1 reduces the transfer time as compared to SACK-IW-LT. For bulk data transfers ER does not offer any advantage as the scenario in which ER can be invoked occurs rarely at end of transfers. Hence SACK-IW-LT-ABC1-ER and SACK-IW-LT-ABC1 overlap in Fig. 6.

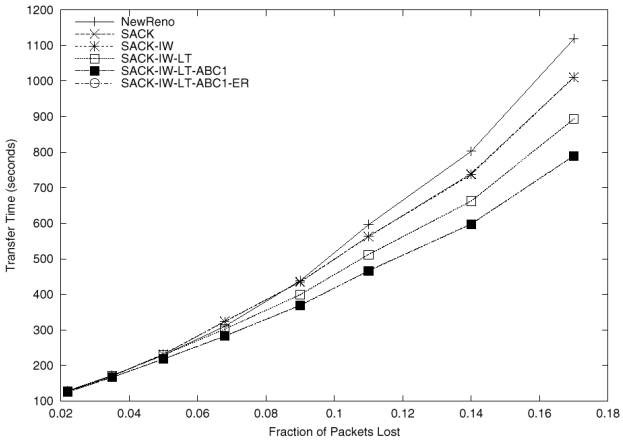


Fig. 6. Transfer Time vs. Fraction of Packet Loss for a 10M transfer

We plot the CDF of the number of times LT and ER get invoked for the bulk transfer vs. the fraction of total number of runs. It is evident from Fig. 7 that the effect of LT is pronounced on bulk transfers where approximately 50% of the total runs invoked LT more than 400 times in their transfers. More than 90% of the runs invoked LT 100 times in their transfers. While the effect of ER on transfer time is minimal, we still saw 2.5% of the runs witness one ER event at the end of transfer. ER may be able to save one timeout for 2.5% of the runs, although a single timeout at the end of the transfer is insignificant relative to the bulk data transfer time.

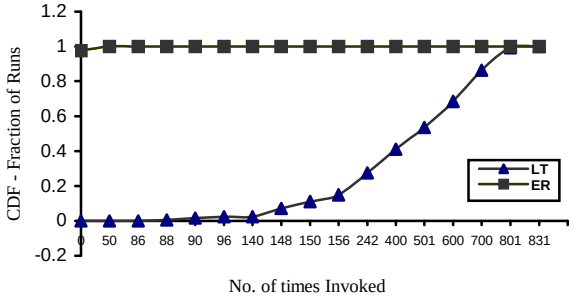


Fig. 7. CDF of the Number of times Limited Transmit or Early Retransmit get invoked vs. Fraction of Runs for a 100K transfer.

V. Conclusions And Ongoing Work

This paper presents measurement and evaluation results for a set of five recent TCP enhancements: SACK and the loss recovery algorithm, Increasing Initial Congestion Window, Limited Transmit, Appropriate Byte Counting and Early Retransmit. We developed TBIT extensions and performed measurements to determine the support of these TCP enhancements in the web server space in the Internet. Our findings were that out of the servers that returned results:

- Majority of the web servers use the standard initial window of two packets. However, 15% of the web servers still use a low initial window of one packet. Only 3% of the servers use an initial window of 4 packets as allowed by [AFP+02].

- While a large number of servers advertise the SACK option, only 30% of them use the SACK information to perform loss recovery.
- More than 50% servers do not support Limited Transmit.
- 50% of the web servers support Appropriate Byte Counting, while other 50% are still vulnerable to misbehaving "split-acking" receivers.
- None of the servers support Early Retransmit.

We performed simulations to assess the performance benefits of these TCP enhancements for web and bulk data transfers. Based on the results obtained, we make the following observations:

- SACK does not offer any noticeable improvements in the transfer time for web transfers. For bulk transfers, SACK reduces the transfer time for loss rates above 10%.
- Increasing initial congestion window improves transfer times for small web transfers but offers no advantage for large web transfers and bulk transfers.
- Although a relatively small change to TCP congestion control, Limited Transmit is an important feature that improves transfer times and gets invoked for all categories of transfers and proportionally to the size of the transfer.
- Appropriate Byte Counting offers protection against misbehaving receivers. From our results it also improves the transfer time significantly for bulk transfers and for large web transfers, in the presence of delayed Acks or Ack loss.
- Early Retransmit is a feature that gets invoked consistently for 3% of the total TCP runs. The main benefits of Early Retransmit occur only for short flows such as small web transfers.

Ongoing work as part of this paper includes the following:

- We are currently running the TBIT tests described in this paper on a list of 27000 web servers. We plan to perform fingerprinting of the operating systems being used by the remote servers, and survey the current implementation status of the TCP enhancements in the popular operating systems.
- The TCP-friendly equation [PFT+98], which is the basis for TCP-friendly rate control (TFRC) [HFP+03], is based on the Reno variant of TCP. We are investigating if the TCP-friendly equation is still a correct representative of current TCP, i.e., TCP with the enhancements evaluated in this paper.

REFERENCES

[AAA+03] M. Allman, K. Avrachenkov, U. Ayesta, J. Blanton, *Early Retransmit for TCP and SCTP*. draft-allman-tcp-early-rexm-03, Dec 03.

[ABF01] M. Allman, H. Balakrishnan, S. Floyd, *Enhancing TCP's Loss Recovery Using Limited Transmit*. RFC 3042, Jan 01.

[AF99] M. Allman, A. Falk, *On the Effective Evaluation of TCP*. ACM Computer Communication Review, 29(5), Oct 99.

[AFP+02] M. Allman, S. Floyd, C. Partridge, *Increasing TCP's Initial Window*. RFC3390, Oct 02.

[All98] M. Allman. *On the Generation and Use of TCP Acknowledgments*. ACM Computer Communication Review, 28(5), Oct 98

[All03] M. Allman, *TCP Congestion Control with Appropriate Byte Counting (ABC)*. RFC 3465, Feb 03.

[ALEXA] Global Top 500 Web Sites. http://www.alexa.com.

[APS99] M. Allman, V. Paxson, W. Stevens. *TCP Congestion Control*. RFC 2581, Apr 99.

[BAF+03] E. Blanton, M. Allman, K. Fall, L. Wang, *A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP*. RFC3517, Apr 03.

[Bal98] H. Balakrishnan, *Challenges to Reliable Data Transport over Heterogeneous Wireless Networks*. Ph.D. Thesis, University of California at Berkeley, Aug 98.

[CAIDA] CAIDA: *Packet Sizes and Sequencing*, Mar 98. http://traffic.caida.org.

[CMT98] K. Claffy, G. Miller, and K. Thompson, *The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone*. INET 98, Apr 98.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, *Hypertext Transfer Protocol -- HTTP/1.1*. RFC2616, Jun 99.

[FF96] K. Fall, S. Floyd, *Simulation-based Comparisons of Tahoe, Reno and SACK TCP*. ACM Computer Communication Review, July 96.

[FH99] S. Floyd, T. Henderson, *The NewReno Modification to TCP's Fast Recovery Algorithm*. RFC 2582, Apr 99.

[HFP+03] M. Handley, S. Floyd, J. Padhye, J. Widmer, *TCP Friendly Rate Control (TFRC) Protocol Specification*. RFC3448, Jan 03.

[JK88] V. Jacobson, M. Karels, *Congestion Avoidance and Control*. In Proceedings of the Sigcomm 1988, Aug 88

[LTW+93] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. *On the Self-similar Nature of Ethernet Traffic*. In *ACM SIGCOMM 1993*, Sep 93.

[MC00] S. McCreary, K. Claffy, *Trends in Wide Area IP Traffic Patterns - A View from Ames Internet Exchange*. Proc. ITC, September 2000. Monterey, CA.

[MJ92] S. McCanne, V. Jacobson, *The BSD Packet Filter: A New architecture for User level Data Capture*. In Proceedings of 1993 Winter USENIX Conference, Jan 93.

[MMF+96] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, *TCP Selective Acknowledgement Options*. RFC2018, Oct 96.

[NMAP] The NMAP Security Scanner. http://www.insecure.org/nmap.

[NS] The Network Simulator-2. http://www.isi.edu/nsnam.

[PF01] J. Padhye, S. Floyd, *Identifying the TCP Behavior of Web Servers*. In Proceedings of Sigcomm 2001, Jun 01.

[PFT+98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. *Modeling TCP throughput: a simple model and its empirical validation*. ACM Sigcomm 98, Sep 1998

[PN98] K. Poduri, K. Nichols, *Simulation Studies of Increased Initial TCP Window Size*, RFC 2415, Sep 98.

[SCW+99] S. Savage, N. Cardwell, D. Wetherall, T. Anderson, *TCP Congestion Control with a Misbehaving Receiver*. ACM Computer Communications Review, 29(5): 71-78, October 1999.

[TBIT] The TCP Behavior Inference Tool (TBIT). http://www.icir.org/tbit.