TCP-DCR: Making TCP Robust to Non-congestion Events* **

Sumitha Bhandarkar and A.L. Narasimha Reddy

Dept. of Electrical Engineering
Texas A & M University
{sumitha,reddy}@ee.tamu.edu

Abstract. In this paper, we propose and evaluate TCP-DCR. TCP-DCR makes simple modifications to the TCP congestion control algorithm to make it more robust to non-congestion events. The key idea here is to delay the congestion response of TCP for a short interval of time τ, thereby creating room for local recovery mechanisms to handle any non-congestion events that may have occurred. If at the end of the delay τ, the event is not handled, then it is treated as a congestion loss. We evaluate TCP-DCR through analysis and simulations. The evaluation is done for three scenarios - a wireless network with channel errors, a wired network with packet reordering and a network with zero non-congestion events. The simulation results show that significant performance improvements can be achieved by using TCP-DCR in the presence of non-congestion events with zero or marginal impact in the absence of non-congestion events. TCP-DCR remains fair to the native implementations of TCP that respond to congestion immediately after receiving three dupacks. TCP-DCR is a simple, effective scheme providing a unified solution to several problems with minimal implementation overhead.

1 Introduction

The strength of TCP lies in the fact that it tries to mitigate congestion in the network by reducing the sending rate in response to loss of packets. Historically, using packet loss as a measure for perceiving congestion has worked quite well. But in the recent past, the nature of the networks has changed significantly. As a consequence, severe penalty is paid in terms of degraded performance in networks where the reason for packet loss is not necessarily network congestion. Recent studies [1], [2] have shown that packet reordering is more prevalent in the current Internet than was assumed earlier rendering the wait of three dupacks used in TCP, an inefficient heuristic. While this in itself is a good reason for investigating the robustness of TCP to non-congestion events such as packet reordering, the authors of [3] present a more compelling reason - the taboo against packet reordering prevents or restricts the research and deployment of several new, beneficial schemes on the Internet for providing efficient routing or differentiated services. Another

* This work is supported in part by a grant from The Texas Higher Education Board, by NSF grant ANI-0087372 and by Intel Corp.

** An extended version of this paper is available in [22]

N. Mitrou et al. (Eds.): NETWORKING 2004, LNCS 3042, pp. 712–724, 2004.

common situation that has spurred the interest in improving the robustness of TCP to non-congestion events is the ever increasing use of wireless networks. Wireless networks are characterized by higher channel error rates than wired networks. When TCP is used in wireless networks, the losses due to channel errors (non-congestion events) are mistaken for congestion losses and the sending rate is unnecessarily reduced, resulting in degraded performance [7]. Several different solutions have been proposed to improve the performance of TCP in the face of packet reordering or in wireless networks. In this paper, we aim to provide a single generalized solution that can be used to improve the robustness of TCP to all non-congestion events.

Our solution is intuitive and employs two simple ideas: (a) delay the congestion response of TCP for a short interval of time τ, creating room to handle any non-congestion events that may have occurred, and (b) employ "local recovery" techniques to recover from non-congestion events during this interval. If at the end of the delay τ the event has not been handled, then it is treated as a congestion event. This simple concept fits into the general philosophy of segregation between the different layers of the network model. The modifications to TCP do not handle the non-congestion event, but rather, rely on some lower layer mechanism to do local recovery, if necessary. To distinguish this flavor of TCP from the original, we call it the Delayed Congestion Response TCP (TCP-DCR for short). This is a general solution that can be extended to any network with non-congestion events and an underlying mechanism for recovering from them.

The rest of the paper is organized as follows. Section 2 provides intuition, analyses and discussion of the TCP-DCR modifications in general. This is followed by discussion and ns-2 [16] simulation results for three specific cases - wired networks with non-negligible packet reordering (Section 3), wireless networks with non-negligible channel errors (Section 4) and finally regular networks with no non-congestion events at all (Section 5). Section 6 concludes the paper by summarising the results and looking at the future work.

2 Delayed Congestion Response TCP

When a TCP receiver finds an intermediate packet missing, but subsequent packets are being received, it sends dupacks to the sender. The sender using the standard TCP algorithms treats the receipt of three consecutive dupacks as an indication that the intermediate packet is lost and responds by reducing the congestion window and triggering the fast retransmit/recovery algorithms. The sender using TCP-DCR is modified to wait for an interval of τ after receiving the first dupack to make room for local recovery of the packet, if possible. The study presented in [17] has shown that even in dynamic network conditions, slowly responding protocols are fair and safe for deployment. In this section we present the details of the proposed TCP-DCR modifications.

2.1 Choice of τ

The delay in responding to congestion determines the performance of TCP-DCR and the choice of τ is a critical aspect for the TCP-DCR modifications. Too large a delay would mean that the protocol responds too sluggishly to congestion in the network. Too small a

delay would not allow the lower layer sufficient time to recover from the non-congestion events. In this section we provide guidelines for choosing reasonable bounds on the delay τ.

Consider first the wireless scenario. Fig. 1 shows a general case where the TCP receiver is connected to a base station over a wireless link. In this scenario, for the link layer to recover a packet lost due to channel error, τ should be atleast as large as the round trip time of the wireless link. On the other hand, to avoid an expensive timeout at the TCP-DCR sender, τ should be smaller than the retransmission timer (RTO) value. The RTO is usually set to RTT + 4 times the RTT variance, where RTT is the estimated end-to-end round trip time. A choice of one RTT for the value of τ allows the link layer sufficient time to recover the packet while at the same time avoiding an RTO. Same argument holds good even when the sender and receiver are connected directly over a wireless link, as in the case of an adhoc network.

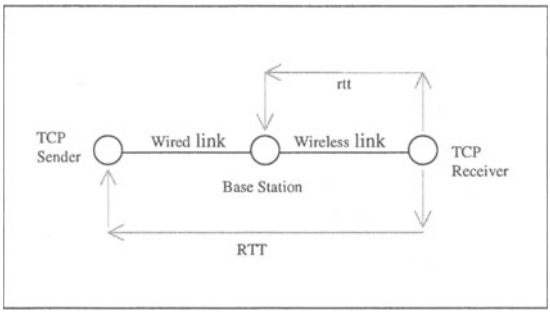

Fig. 1. Analysis of TCP-DCR in a Wireless Network with no Congestion Losses

In the case of packet reordering, the amount by which the packet is reordered could be highly variable - the time to recover the lost packet is the time that the reordered packet takes to reach the receiver. Hence there is no preset lower bound for the delay τ, that will facilitate the recovery of all reordered packets. However, the upper bound is still decided by the value of the RTO. So, a value of one RTT for τ is still a reasonable choice.

Based on the discussion above, we modify the heuristic for the wait after the sender starts receiving dupacks (τ) to one RTT. Setting τ to one RTT, rather than a fixed value, also provides inherent robustness to fluctuations in the queuing delays ensuring that we do not get into RTO timeout even during sudden changes in the network load.

2.2 Steady State Analysis of TCP-DCR

The steady state throughput of TCP-DCR with the assumption of uniform periodic loss probability model can be shown to be similar to that of TCP ($throughput \propto \frac{1}{\sqrt{p}}$). Detailed analysis has been omitted here due to lack of space. An interested reader may

find the same in [22]. The main difference however, is that in the presence of both non-congestion events and congestion losses, for the standard TCP algorithm p is the sum of the non-congestion event rate and the loss rate due to congestion, but for TCP-DCR p is only the loss rate due to congestion. As a result, in the presence of both non-congestion events and congestion, TCP-DCR can achieve better throughput.

2.3 Receiver Buffer Requirement When TCP-DCR Is Used

When TCP-DCR is used, the receiver will need to have additional buffer space to accommodate the extra packets corresponding to the delay τ, when a packet is lost due to congestion. Having these extra buffers allows TCP-DCR to achieve the best performance. In the absence of the additional buffer space, the flow control mechanism of TCP limits the sending rate, preventing TCP-DCR from achieving the maximum performance improvement.

2.4 Local Recovery Mechanisms

The performance benefits to be gained from using the TCP-DCR modifications depend on the existence of an underlying scheme for recovering the losses due to non-congestion events. In case of packet reordering, nothing needs to be done explicitly to recover the reordered packet. In case of wireless networks, we assume that the underlying mechanism is a simple link level retransmission scheme, possibly NACK-based, that does not attempt in-order delivery. Some of the recent research in the area of networking for multimedia [18] also advocate the use of link level retransmission schemes that do not attempt in-order delivery. Alternatively, FEC (Forward Error Correction) schemes could also be used.

2.5 Summary of Modifications

The TCP-DCR modifications need to be applied only to the sender. The congestion response is delayed only during the congestion avoidance phase. During the congestion response delay, the congestion window continues to evolve using additive increase but only one new packet is transmitted in for each dupack (similar to the proposed standard limited transmit algorithm [15]). Thus, TCP-DCR remains ack-clocked during the congestion response delay period and the sending rate during τ remains at best, the same as when the first dupack was received.

If the congestion response delay timer expires, the fast retransmit/recovery algorithms are triggered. The *ssthresh* and the congestion window are set to half the current value of the congestion window just as it would be in a traditional implementation of TCP.

The sender can implement the delay either by using a timer or by modifying the threshold on the number of dupacks to be received before triggering the congestion recovery algorithms (*dupthresh*). The timer based implementation however depends on the clock granularity. To ensure that faulty implementation of the timer does not result in an RTO, for the timer-implementation the timer should be set to one RTT as indicated by

the *smoothed_rtt* and for the dupack-based implementation, the new value for *dupthresh* should be scaled by the factor $(smoothed\_rtt)/(current\_instantaneous\_rtt)$.

The TCP-DCR modifications work with most flavors of the TCP protocol. However, in this paper we advocate the use of TCP-DCR with TCP-SACK [14], if the TCP-SACK option is available. When used with TCP-SACK, the only thing modified by TCP-DCR is the time at which the fast retransmit/recovery algorithm is triggered in response to the first loss within a window of packets. All subsequent losses within the same window (irrespective of whether they are due to congestion or non-congestion events) are handled in exactly the same way as TCP-SACK would in the absence of TCP-DCR modifications.

Use of delayed_acks will not intervene with the TCP-DCR modifications, provided that the implementation of delayed acks follow the guidelines in [19] that the dupacks (or SACKs) are not delayed.

3 TCP-DCR and Reordering Robustness

In current networks, packet reordering is observed to be not negligible [1],[2]. Also, many new design alternatives for routers or network architectures may benefit if there are no strict restrictions of zero packet reordering. Several different solutions have been proposed in literature to solve this problem. In [3] and [4] the authors present schemes for improving the reordering robustness of TCP that use DSACKs [5] or timestamps [6] to identify reordering and the possible amount of reordering. TCP-DCR on the other hand, aims to improves the reordering robustness of TCP without having to identify the exact amount of reordering in the network or using complex state or algorithms.

3.1 Simulation Topology

We evaluated the performance of TCP-DCR using the ns-2 simulator [16] (version 2.26). A simple dumbbell topology is used where n different sources are connected to n different receivers via a single bottleneck link between routers R1 and R2. The default values for the bandwidth and delay for the links between the routers and the end nodes is fixed at 10 Mbps and 1 ms respectively. The bandwidth and the delay for the bottleneck link is varied in accordance with the requirements of the experiment. Each source i performs bulk data transfer to the receiver i with a packet size of 1000 bytes. DropTail buffer management scheme is used at the routers and the queue size is set to 50 packets, unless otherwise specified. Packet reordering is simulated by modifying the errormodel object of ns-2 such that randomly selected packets can be delayed for a random amount of time.

The TCP-DCR agent is implemented by modifying the tcp-sack1 implementation of TCP-SACK agent in ns-2. Ack-based implementation is used for the congestion response delay. The TCPSink/Sack1 agent is used for the receivers. FTP sources start sending data at time 0 and are staggered to avoid synchronization. All simulations are run for 1100 seconds, but data is collected only after the first 100 seconds to ensure that steady state is reached. The receiver advertises a large window such that the sending rate is not limited by the receiver dynamics.

3.2 Performance at Varying Packet Delay Rate

One of the primary reasons for reordering in the network is that some of the packets get delayed more than others, and hence arrive out of order. In this experiment randomly selected packets are delayed, with a delay picked from a normal distribution with a mean of 25ms and a standard deviation of 8ms so that most chosen packets are delayed in the range 0 to 50ms. This simulates mild but persistent reordering. The bottleneck link bandwidth is set to 8Mbps and the delay to 50ms. There is no congestion in the network. The topology consists of a single flow. The experiment is first run with TCP-SACK and repeated for TCP-DCR. Fig. 2 shows the results

As can be seen from the graph, the performance of TCP-SACK degrades rapidly, since persistent reordering keeps the sender congestion window small, reducing the throughput drastically. TCP-DCR performs significantly better than TCP-SACK. Since there is no congestion in the network, and the packets are only mildly reordered, most packets are recovered during the delay in the congestion response.

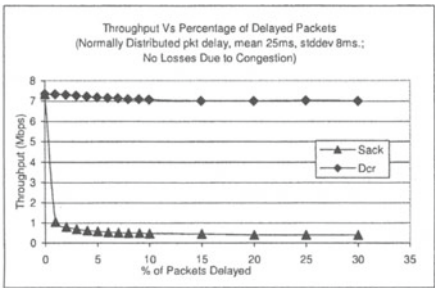

Fig. 2. Throughput Vs Percentage of Packets Delayed (With Single Flow)

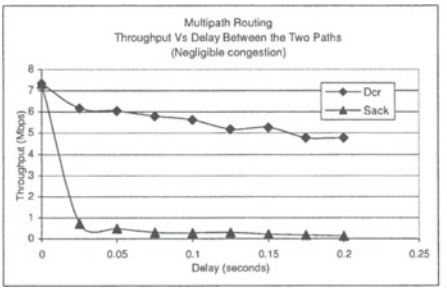

Fig. 3. Performance Comparison with Multi-path Routing

3.3 Performance Comparison with Multi-path Routing

One of the situations that can cause packet reordering is when packets are routed over different paths. Suppose, a router chooses between two different paths for load balancing. In the worst case, alternate packets get routed over the different routes, causing 50% of all packets to get delayed. In this simulation we examine such a situation. The x-axis shows the difference between the RTTs of the two routes. The link delay of the shorter route is fixed at 50ms. Fig. 3 shows the results. It can be seen from the graph that TCP-DCR performs significantly better than TCP-SACK. When the delay between the two paths becomes larger than the round trip time of the shorter path, the performance of TCP-DCR starts to degrade a little. However, the smoothed RTT estimate at the TCP sender will reflect the average round trip time of the link, and the congestion response delay is scaled by this value. As a result, the performance degradation is not drastic.

3.4 Performance Comparison with Congestion in the Network

One of the primary concerns with using TCP-DCR is the effect of delaying congestion response on other flows in the network. We study that in this experiment. The bottleneck link has a capacity of 10Mbps and a link delay of 10ms. The number of flows in the network is 12, with 6 of them using TCP-DCR and the other 6 using TCP-SACK. Congestion in the network is controlled by varying the buffer size at the router R1 for the link between R1-R2. Fig. 4 shows the results when 10% of the packets are delayed. Thus, packets are reordered as well as lost due to congestion in this simulation. When a packet is lost due to congestion, the sending rate is reduced both in the case of TCP-SACK and TCP-DCR. However, when a packet is reordered, the sending rate is reduced only in the case of TCP-SACK. As a result at low congestion levels, TCP-DCR flows utilize more link capacity than TCP-SACK flows and show better throughputs. When the congestion levels in the network increases, the link capacity is more and more equitably shared. It is to be noted that the reason for TCP-DCR realizing better throughputs (when packets are reordered) is not due to unfairness, but due to correctly recovering from the reordering events (without reducing the congestion window). We address the fairness issue in section 5 when we consider zero non-congestion events.

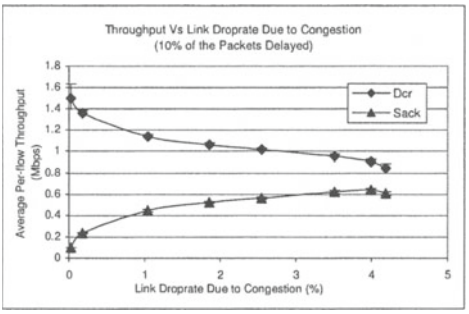

Fig. 4. Throughput Vs Link Droprate due to Congestion

4 TCP-DCR in a Wireless Network

Wireless networks are characterized by high channel error rates. When TCP is used in wireless networks, the losses due to channel errors are mistaken for congestion losses and the sending rate is unnecessarily reduced, resulting in degraded performance [7]. Several solutions have been proposed to improve the performance of TCP over wireless networks. These solutions fall in one of the following broad categories: (a) Split connection approaches (eg. [8]) (b) TCP-aware link layer protocols (eg. [9]) (c) Explicit loss notification approaches (eg. [10]) (d) Receiver-based approaches (eg. [11]) and (e) Modifications to TCP (eg. [12]) When both congestion losses and losses due to the transmission errors can occur, a simple solution would be to let the link layer mechanisms to recover from losses due to transmission errors, allowing the transport protocol to recover

from congestion losses. When TCP-DCR is used in wireless networks, a simple link level retransmission scheme that is not aware of TCP semantics would suffice to recover from transmission errors without any explicit notification from the network regarding the type of the loss. Earlier work has shown that local recovery of channel errors is efficient [13].

4.1 Simulation Topology

The network topology used in these simulations is similar to that in the previous section, except that R2 is the Base station connected to the receivers via wireless links. The default values for the wired link bandwidth and delay is fixed at 100 Mbps and 5 ms respectively. The wireless link bandwidth is kept fixed at 1 Mbps and the delay is varied in accordance with the requirements of the experiment.

The TCP-DCR agent in these simulations uses the timer-based implementation of the congestion response delay. Link level retransmission is simulated by using the error model and the queue object provided by ns-2. The error model is exponential, and the corrupted packets are buffered at the base station and retransmitted after a delay corresponding to the round trip time of the wireless link, thus simulating link level retransmission. The packet to be retransmitted is added at the head of the queue that holds the packets awaiting transmission. The TCP/IP and MAC layer headers are ignored in the throughput calculations.

4.2 Performance at Different Channel Error Rates

First, we present the results for the simulation showing the performance improvement offered by TCP-DCR at various channel error rates in Fig. 5. The workload consists of a single flow in this case. The wireless link bandwidth and delay are set to 1Mbps and 45ms respectively, so that the total round trip time is comparable to the simulation in section 3.2. There is no congestion in the network. As can be seen from the graph, TCP-DCR performs better than TCP-SACK. Since there is no congestion in the network, most of the packet losses can be recovered using the link layer retransmission scheme making a window reduction unnecessary. Thus, the performance of TCP-DCR even at high channel error rates stays close to the performance that can be obtained when there is no channel errors at all. On the other hand, TCP-SACK treats the losses due to channel errors as congestion loss and hence the throughput stays below 0.5 Mbps.

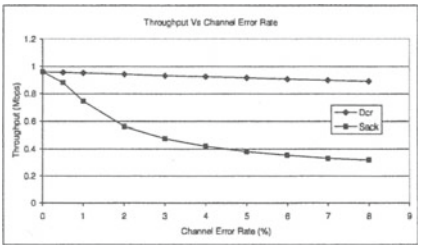

Fig. 5. Throughput Vs Channel Error Rate

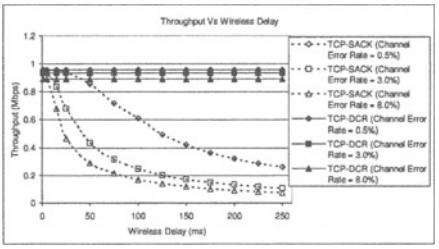

Fig. 6. Throughput Vs Wireless Link Delay

4.3 Performance at Different Wireless Delays

Wireless networks have highly varying delays ranging from few milliseconds to few tens of milliseconds for a LAN to several hundred of milliseconds for satellite links[20,21]. In this section we show the effect of the wireless delay on the performance of the different protocol flavors. The topology is similar to that in the previous section. Fig. 6 shows the results. It can be seen from the graph that as the wireless link delay is increased, the throughput of the TCP-SACK flows degrades significantly. This is because when the window is reduced incorrectly due to a packet lost by channel errors, it takes a longer time for the protocol to increase the window to the correct value again.

4.4 Performance with Congestion in the Network

In this set of simulations, the workload consists of 24 flows, half of which use TCP-DCR and the other half use TCP-SACK. The different levels of congestion are obtained by varying the buffersize at the router R1. The bottleneck link capacity is set to 10Mbps and the delay to 5ms. The wireless link bandwidth and delay are 1Mbps and 20ms. Fig. 7 shows the results. In the graph, congestion loss rates of less than 1% are labelled as low error, in the range of 2.5-3.5% are labelled as moderate congestion and greater than 3.5% are labelled as high congestion.

It can be seen from the figure that when the congestion loss rate is low, the average throughput of the TCP-DCR flows is far more than that of TCP-SACK flows. The throughput achieved by TCP-DCR flows is inversely proportional to the congestion loss rate in the network, whereas the throughput of the TCP-SACK flows is inversely proportional to the sum of the congestion loss rate and the channel error rate. So, as the congestion loss rate in the network increases, the difference in the average throughput of the TCP-DCR flows in the network compared to that of the TCP-SACK flows becomes narrower.

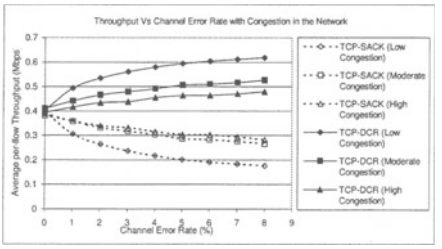

Fig. 7. Throughput Vs Channel Error Rate with Congestion in the Network

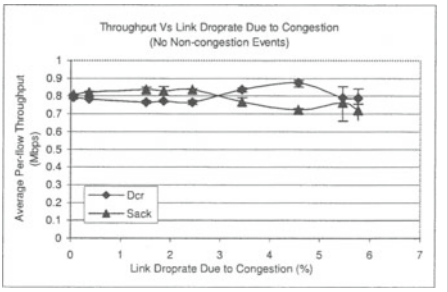

Fig. 8. Throughput comparison with Zero Non-congestion Events.

5 TCP-DCR with Zero Non-congestion Events

The earlier two sections have shown that TCP-DCR provides a simple, but effective mechanism for tolerating non-congestion events in networks that cause packet reordering or have significant channel errors. The natural questions that arise: what is the consequence of employing TCP-DCR in networks that do not experience any non-congestion events? Does TCP-DCR impact the throughput realized by individual flows? Is it fair to other flows that respond to congestion immediately? Does it impact queue lengths? We examine such question in this section.

5.1 Fairness

The results in section 3.4 and section 4.4 show that in the presence of non-congestion events, TCP-DCR utilizes the network bandwidth better than TCP-SACK flows at lower congestion, and the bandwidth is shared more equitably as congestion losses become the major contributing factor towards the total losses. In this section we evaluate the fairness of TCP-DCR when there are no non-congestion events at all in the network.

The simulation set up is similar to that in section 3.4. The graph shows the average throughput realized by DCR and SACK flows. From Fig. 8 we see that the average throughput achieved by the DCR flows is very close to the average throughput of the SACK flows, even at fairly high levels of congestion. The throughput of each individual flow does not vary too much from the average as indicated by the confidence intervals. These results indicate that TCP-DCR does not behave more aggressively than TCP-SACK, when τ is set to one RTT.

5.2 Packet Delivery Time and RTT Estimates

Since TCP-DCR delays the congestion response by one RTT, it takes a longer time to recover a packet lost due to congestion when compared to SACK. In order to evaluate the extent of the additional time taken by DCR, we conducted this experiment. For the network topology chosen, majority of the packets for both the flavors are delivered in 0.05seconds. For packets lost due to congestion and recovered using TCP retransmission, the average packet delivery time for case 1 (100% flows use TCP-DCR) is 207ms, for case 2 (100% flows use TCP-SACK) it is 178ms and for case 3 (50% of the flows use TCP-DCR and 50% use TCP-SACK), it is 201ms for TCP-DCR and 182ms for TCP-SACK. TCP-DCR does not affect the packet delivery time when there is no congestion. However, when a packet is lost due to congestion, the time to recover it could be higher by about one RTT. Also, according to Karn's algorithm used by most standard implementations of TCP, a retransmitted packet is not used in estimating the round trip time. Thus the delayed congestion response of TCP-DCR does not affect the rtt estimation of TCP. Our simulation results agree with the discussion above. An interested reader may find the detailed results in [22].

5.3 Response to Sudden Increase in Traffic

In this experiment we study the response of TCP-DCR to sudden increase in the traffic on the network. For this experiment, we first allowed six flows to run for 50 seconds until

they reached steady state. At the end of 50 seconds, an additional six TCP-SACK flows were added. We compared the response of TCP-DCR with that of TCP-SACK for this sudden increase in traffic. Fig. 9 shows the results. It can be seen from the graph that the response of TCP-DCR is similar to that of TCP-SACK. The time to reach (55%, 45%) allocation for TCP-SACK was 3.1 seconds and for TCP-DCR, it was 3.67 seconds.

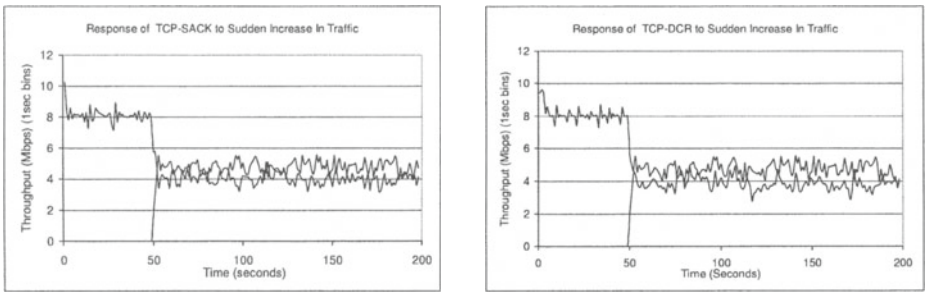

Fig. 9. Response to Sudden Increase in Traffic

5.4 Summary of Other Observations

We conducted several other experiments to evaluate the queue lengths, the timeouts, the perflow droprates, and the link utilization to understand the impact of DCR flows on the network characteristics. We summarize these results for three cases - case 1 (100% flows use TCP-DCR), case 2 (100% flows use TCP-SACK) and case 3 (50% of the flows use TCP-DCR and 50% use TCP-SACK) - in Table 10. As seen from these results, DCR flows do not drastically alter the observed network characteristics compared to a network with only SACK flows.

Case	Protocol of flow	Avg.Flow throughput Mbps	Avg.Flow Drop rate %	Avg.Flow timeouts %	Avg.que Length Pkts
1	DCR	0.801	2.72	0.0	44
2	SACK	0.801	2.27	0.009	44
3	DCR	0.748	2.77	0.0	45
	SACK	0.852	2.13	0.02	

Fig. 10. Summary of observations with Zero Non-congestion Events.

We also conducted several simulations with the sudden increase in background traffic due to several short term TCP flows simulating web-traffic. The results were similar to those in Fig. 9.

6 Conclusions and Future Work

In this paper, we proposed TCP-DCR that employs delayed congestion response and local recovery to recover from non-congestion events. We studied DCR's handling of non-congestion events in two specific scenarios, namely, packet reordering and wireless channel errors. In both the scenarios, results from simulations have shown that DCR offers significantly better performance by simply delaying congestion response for one RTT. We then studied the impact of employing DCR in networks with zero non-congestion events. Several other simulations were conducted the results of which have not been included here due to lack of space. An interested reader can find them in [22]. Our evaluation at multiple levels - individual flows, TCP characteristics and network characteristics - has shown that DCR does not significantly impact other flows or the network even when all the packet losses are due to congestion alone. Based on these results, DCR seems to offer a simple, unified solution to handle non-congestion events safely. We have also implemented DCR on a Linux platform and the preliminary results (available in [22]) look promising. We plan to continue with the tests under different scenarios.

Acknowledgements. Nauzad Sadry and Nitin Vaidya have contributed to the work reported in the wireless network section. Comments from Sally Floyd on an earlier draft have helped the paper.

References

1. Jon Bennett, Craig Partridge, and Nicholas Shectman, "Packet Reordering is Not Pathological Network Behavior," *IEEE/ACM Transactions on Networking*, December 1999.
2. Sharad Jaiswal, Gianluca Iannaccone, Christophe Diot, Jim Kurose, and Don Towsley, "Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP Backbone," *Proceedings of IEEE INFOCOM*, 2003.
3. M. Zhang, B. Karp, S. Floyd, and L. Peterson, "RR-TCP: A Reordering-Robust TCP with DSACK," *ICSI Technical Report TR-02-006*, Berkeley, CA, July 2002.
4. E. Blanton and M. Allman, "On Making TCP More Robust to Packet Reordering," *ACM Computer Communication Review*, January 2002.
5. Sally Floyd, Jamshid Mahdavi, Matt Mathis and Matt Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP," *RFC 2883*, July 2000.
6. R. Ludwig and M. Meyer, "The Eifel Detection Algorithm for TCP," *RFC 3522*, April 2003.
7. H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *IEEE/ACM Transactions on Networking*, 1997.
8. K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *ACM Computer Communications Review, vol. 27, no. 5*, 1997.
9. H. Balakrishnan, S. Seshan, E. Amir and R. Katz, "Improving TCP/IP performance over wireless networks," *Proc. of ACM MOBICOM*, Nov. 1995.
10. H. Balakrishnan and R. H. Katz, "Explicit Loss Notification and Wireless Web Performance," *Proc. of IEEE GLOBECOM*, Nov. 1998.

11. N. H. Vaidya, M. Mehta, C. Perkins and G. Montenegro, "Delayed Duplicate Acknowledgement: a TCP-unaware Approach to Improve Performance of TCP over Wireless," *Journal of Wireless Communications and Mobile Computing*, special issue on Reliable Transport Protocols for Mobile Computing, February 2002.
12. S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links ," *Proceedings of ACM MOBICOM*, 2001.
13. D. Eckhardt and P. Steenkiste, "Improving Wireless LAN Performance via Adaptive Local Error Control," *Proceedings of IEEE ICNP*, Austin, TX, 1998.
14. M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, "TCP selective acknowledgement options," *Internet RFC 2018.*
15. M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit," *RFC 3042, Proposed Standard*, January 2001.
16. ns-2 Network Simulator. **http://www.isi.edu/nsnam/**
17. D. Bansal, H. Balakrishnan, S. Floyd and Scott Shenker, "Dynamic Behavior of Slowly Responsive Congestion Control Algorithms," *Proceedings of ACM SIGCOMM*, Sep. 2001.
18. R. Han and D.G. Messerschmitt, "A Progressively Reliable Transport Protocol For Interactive Wireless Multimedia", *ACM/Springer-Verlag Multimedia Systems Journal, vol. 7, no. 2,* March 199 9.
19. M. Allman, V. Paxson and W. Stevens, "TCP Congestion Control," *RFC 2581*, April 1999.
20. M. Allman, D. Glover and L. Sanchez, "Enhancing TCP Over Satellite Channels using Standard Mechanisms," *RFC 2488*, January 1999.
21. J. Border, M. Kojo, J. Griner, G. Montenegro and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," *RFC 3135*, June 2001.
22. Sumitha Bhandarkar, Nauzad Sadry, A. L. N. Reddy and Nitin Vaidya, "TCP-DCR: A Novel Protocol for Tolerating Wireless Channel Errors" *Tecnhical Report TAMU-ECE-2003-01*, February 2003.