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, simulations and Linux implementation. The evaluation is done for two scenarios - a wireless network with channel errors and a wired network with packet reordering. The simulation results show that significant performance improvements can be achieved by TCP-DCR in the presence of non-congestion events, with none or marginal impact in the absence of non-congestion events. Also, 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 adjusts the sending rate in accordance with the perceived congestion in the network. The TCP sender uses the acknowledgements from the receiver to gauge the state of congestion in the network. When the receiver receives a packet with higher sequence number than it expects, it sends a duplicate acknowledgement (dupack, for short) for the highest packet received in-order. When the sender receives three consecutive dupacks, it concludes that the packet causing the dupacks is lost due to congestion. It responds by triggering the fast retransmit/recovery algorithms. The fast retransmit algorithms allows a packet to be retransmitted without waiting for a timeout to signal packet loss. The fast recovery algorithm allows the sender to reduce the congestion window by half (instead of reducing it to 1) and as a result allows faster recovery to the original window compared to the recovery after a timeout. The fast retransmit/recovery algorithms perform well when most of the losses are due to congestion. In modern networks, *non-congestion* events can be caused due to wireless channel errors or packet reordering within the network. When the cause for

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

the generation of dupacks is a non-congestion event, the unnecessary reduction in the sending rate results in suboptimal performance.

Recent studies [1], [2] have shown that packet reordering is more prevalent in the current Internet than was assumed earlier, rendering the heuristic of three dupacks insufficient for recovering the out-of-order packets. While this in itself is a good reason for investigating the reordering robustness of TCP, the authors of [3] present a more compelling reason. Excessive reordering can degrade the performance of TCP in the network, and hence any mechanism that may cause reordering is not considered as a design candidate for the Internet (even if it could improve performance or network service). Several new beneficial schemes such as differentiated services and multi-path routing cannot be deployed or are restricted in their utility since they have a potential for causing packet reordering. In the interest of continued research and development in these areas, it is extremely important to ensure that TCP is made more robust to packet reordering in the network. Solutions for packet reordering have been recently investigated in [3],[4].

Wireless networks are characterized by higher channel error rates than in 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 [5]. Several solutions have been proposed to improve the performance of TCP over wireless networks [6-19].

In this paper, we present a general solution for improving the performance of TCP in the presence of non-congestion events. Our solution employs two key 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 short interval, leaving TCP to handle only congestion events after the delay interval. If at the end of the delay τ the event is not handled, then it is treated as a congestion event. The modifications to TCP themselves do not handle the non-congestion event, but rather rely on some underlying mechanism to do local recovery. For instance, in case of reordering due to a packet getting delayed randomly in the network, the delay τ should be chosen long enough to allow the reordered packet to reach the receiver and generate an acknowledgement. In case of wireless networks with channel errors, lost packets can be recovered by a link-level retransmission scheme. To distinguish this flavor of TCP from the original, we call it the Delayed Congestion Response TCP (TCP-DCR for short). While this is a general solution that can be beneficial in any network with non-congestion events and an underlying mechanism for recovering from them, we focus the discussion on the two cases mentioned so far - networks with non-negligible packet reordering and wireless networks with non-negligible channel error rates.

The rest of the paper is organized as follows. In section 2 we present the details of TCP-DCR, analysis of fairness, and some discussion. In section 3, we discuss using TCP-DCR in networks with packet reordering and evaluate it via simulations on the ns-2 simulator [20]. In section 4, we discuss using TCP-DCR in wireless networks and present the results of the simulations in wireless networks. In section 5, we study the impact of delaying congestion response in a network with zero non-congestion events.

In section 6 we provide the preliminary results for experiments on Linux. Section 7 presents the conclusions and the future work.

2 Delayed Congestion Response TCP

When the TCP sender starts to receive dupacks, it waits for three consecutive dupacks before concluding that the packet has been lost due to congestion. This wait of three dupacks is purely heuristic and not based on any scientific evaluation, and is meant to allow mildly reordered packets to be recovered. When the network is responsible for non-negligible amounts of non-congestion events, this wait of three dupacks is very short and drastic. We extend this concept further by allowing the TCP-DCR sender to wait for an interval of τ after receiving the first dupack before responding to it as if it were packet lost due to congestion.

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 link layer sufficient time to recover from the losses due to channel errors. Hence it is essentially a tradeoff between unnecessarily inferring congestion, and unnecessarily waiting for a long time before retransmitting a lost packet and choosing the correct value for the delay is extremely important. In this section we provide guidelines for choosing reasonable bounds on the delay to make it useful, without adversely modifying the TCP behavior. At this point, we note that the current practice of waiting for three dupacks at the sender is merely a heuristic.

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. The wired path between the base station and the TCP sender could consist of several hops, but would not affect the discussion here and so is shown as a single hop. The round trip time between the base station and wireless link is indicated by rtt and the end-to-end round trip time between the TCP sender and the TCP receiver is indicated by RTT.

In the above scenario, if we ignore ambient delays (e.g., inter-packet delay, queuing delay, etc.), a packet sent by the TCP sender at some time $t0$ reaches the base station at $t0 + (RTT/2 - rtt/2)$ and the receiver at time $t0 + RTT/2$. Suppose, a packet k sent at time $t0$ is lost on the wireless link due to channel errors. Then at $t0 + RTT/2 + rtt/2$ the base station receives indication that the packet k is lost. If it immediately retransmits the packet, then the packet k is recovered at the receiver at time $t0 + RTT/2 + rtt$. The sender receives an acknowledgement for the packet k at $t0 + RTT/2 + rtt + RTT/2$. Hence the sender would have to delay the congestion response by at least rtt time units after receiving three DUPACKs, to allow the link layer to recover the packet. In practice, the inter-packet delays are non-zero and the TCP sender may not know the value of rtt. Hence, a simple solution would be to set the lower bound on the delay in congestion response to one RTT.

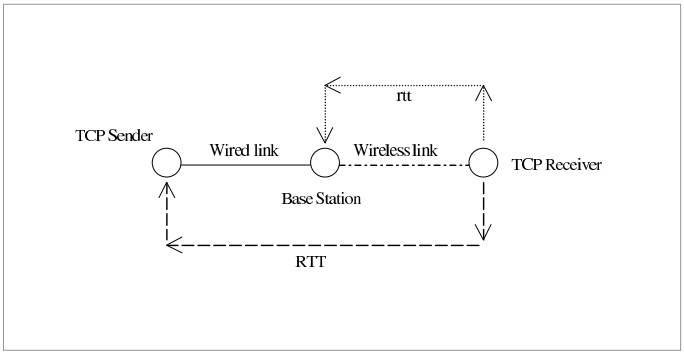

Fig. 1. Analysis of TCP-DCR in a Wireless Network with no Congestion Losses

The TCP protocol uses two mechanisms for identifying congestion in the network - the receipt of three dupacks and the retransmission timeout (RTO). The receipt of three dupacks is considered to be an indication of mild congestion in the network and hence the response to it is the triggering of fast retransmission/recovery algorithms. An RTO, on the other hand is treated as an indication of severe congestion in the network, and so in response to it, the congestion window is reset to 1 and the window evolution starts over with a slowstart. This is an extremely expensive operation. The choice of τ should be such that unnecessary retransmission timeouts are avoided. Thus, the upper bound on the delay τ is imposed by the retransmission timer of TCP. The RTO is usually set to RTT + 4 times the measured variance in RTT. The standard recommends a minimum of 1 second for the RTO, but many TCP implementations have a much smaller minimum, e.g., 100 ms. A choice of one RTT or less for the delay τ, can ensure that RTO can be avoided. Thus, the upper limit on the value of τ is one RTT.

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 discussion above. So, a value of one RTT for tau is still a reasonable choice.

Based on the discussion above, we conclude that a choice of waiting for one RTT after the first dupack before responding to congestion is optimal. By setting the delay to one RTT, rather than a fixed value, we also provide 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 Implementation

The TCP-DCR modifications need to be applied only to the sender. The receiver remains unmodified. The congestion response is delayed only during the congestion avoidance phase and hence does not modify the behavior during the slow start phase. During

the congestion response delay, the congestion window continues to evolve as indicated by the congestion avoidance algorithm (additive increase). However, only one new packet is transmitted in response to each dupack received. This is similar to the proposed standard limited transmit algorithm [21]. This ensures that TCP-DCR remains ack-clocked during the congestion response delay period and a new packet is put on the network only when indication is received that one of the previously sent packets has left the network. Thus the sending rate of the TCP-DCR sender 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 is quite straight forward, but depends on the clock granularity. In the dupack-based delay implementation, the sender could delay responding to congestion for a window of packets, with the window corresponding to the delay required. Thus, when τ is chosen to be one RTT, the sender would wait for the receipt of W dupacks, before responding to congestion, where W is the sending window when the first dupack is received. The implementation of the delay should take care that a faulty implementation does not result in an RTO. So, for the timer-implementation we suggest that the timer be set to one RTT as indicated by the *smoothed_rtt* estimate since the RTO estimate is computed based on the *smoothed_rtt*. In case of the dupack-based implementation, the number of dupacks correspond to the estimate of *current_instantaneous_rtt* and so we suggest that the new value for *dupthresh* 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 to ensure that the performance can be maintained high even under the conditions of multiple losses per round trip time. 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 dupacks generated by the first loss within a window of packets. All subsequent losses within the same window (irrespective of whether they are due to congestion losses or non-congestion events) are handled in exactly the same way as TCP-SACK would in the absence of TCP-DCR modifications. If the receiver is not SACK-capable, however, then the sender will have to use TCP-DCR with other flavors such as NewReno. If several packets are lost in one RTT, then the number of dupacks being received is less, and because of the ack-clocked nature of the sender, it implicitly forces the sender the reduce its sending rate.

Use of delayed_acks will not intervene with the TCP-DCR modifications, provided that the implementation of delayed acks follow the guidelines in [29] that the dupacks (or SACKs) are not delayed.

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. However, if the buffers are not available, it does not degrade the performance drastically, but the maximum performance improvement is not achieved. This is because, apart from congestion control, TCP also provides flow control such that a faster sender does not flood a slow receiver. The flow control is achieved by using a receiver advertised window, such that at any point the TCP sender may not send more packets than that allowed by $min(cwnd, rwnd)$ where $cwnd$ is the congestion window and $rwnd$ is the receiver advertised window. When the buffer space is not available, the receiver advertised window is small. As a result, during the delay τ even though the limited transmit and congestion window allow a packet to be transmitted it will not be sent if the $rwnd$ (and hence the receiver buffer) does not allow it. However, the TCP sender can still delay the congestion response by τ allowing the local recovery mechanism to recover from non-congestion event.

2.4 Steady State Analysis of TCP-DCR

In this section we present an analysis of the steady state bandwidth of TCP-DCR. We use the steady state model with uniform congestion loss probability p.

The congestion window for TCP-DCR can be represented using two functions $f_1(t)$ and $f_2(t)$, where $f_1(t)$ determines the window behavior before the time t_{drop} when a packet is dropped and $f_2(t)$ determines the behavior after the packet drop. The function $f_1(t)$ is the additive increase function. The function $f_2(t)$ has two components. For the time period τ between t_{drop} and $t_{drop+\tau}$, $f_2(t)$ continues with the additive increase function. Immediately after the congestion delay timer expires, i.e., at $t_{drop+\tau+\epsilon}$, the congestion window is decreased multiplicatively. These two functions can be represented as follows-

$$\begin{aligned} f_1(t) \quad & : w_{t+RTT} \leftarrow w_t + \alpha; \alpha > 0 \\ f_2(t) \quad & : w_{t+RTT} \leftarrow w_t + \alpha; \alpha > 0, t_{drop} < t < t_{drop} + \tau \\ & \quad w_{t_{drop+\tau}} \leftarrow \gamma * w_{t_{drop+\tau-\epsilon}}; \gamma > 0, t = t_{drop+\tau} \end{aligned} \tag{1}$$

where w_t is the congestion window at time t, RTT is the round trip time, τ is the delay in congestion response and α and γ are constants. Fig. 2 shows the graphical representation of the congestion window against time.

Let T_D be the time between two successive drops and let N_D be the number of packets sent by the protocol in this time. From equation [1], using continuous fluid approximation and linear interpolation of the window between w_t and w_{t+RTT} we get

$$\frac{dw}{dt} = \frac{\alpha}{RTT} \quad \Rightarrow \quad w = \frac{\alpha t}{RTT} + C \tag{2}$$

As can be seen from Fig. 2, the parameters T_D and N_D are independent from time shifting the curve along the horizontal (time) axis. This implies that one can arrange

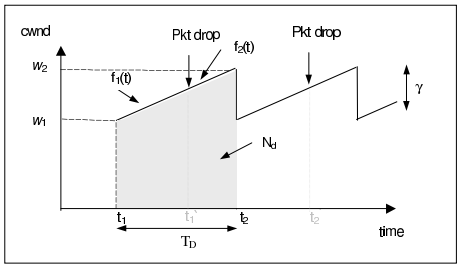

Fig. 2. Analysis of TCP-DCR with no Channel Errors

it such that a downward interpolation of the curve passes through the origin. That is, without loss of generality and with no change to T_D and N_D , one can set C = 0. Thus we have,

$$\begin{aligned} w &= \frac{\alpha t}{RTT} \\ \Rightarrow t &= \frac{wRTT}{\alpha} \end{aligned}$$

The throughput λ (in packets per second) can be given by the number of packets that can be sent between two successive drops (N_D) divided by the time interval between two successive drops (T_D). From the Fig. 2 we have,

$$\begin{aligned} T_D &= t_2' - t_1' = t_2 - t_1 \\ &= \frac{RTT}{\alpha}(w_2 - w_1) \end{aligned}$$

The window reduction is determined by the constant γ. Hence we have, $w_1 = \gamma w_2$. Substituting this in the above equation, we get,

$$T_D = \frac{RTT}{\alpha} \cdot w_2 \cdot (1-\gamma) \tag{3}$$

N_D is the shaded are under the curve in Fig2. Hence,

$$N_D = \int_{t_1}^{t_2} w(t)\frac{dt}{RTT} = \frac{1}{2\alpha} \cdot w_2^2 \cdot \left(1-\gamma^2\right) \tag{4}$$

However, since N_D is the number of packets between two consecutive drops, the steady state drop probability p = 1/N_D.

$$\begin{aligned} \frac{1}{p} &= N_D = \frac{1}{2\alpha} \cdot w_2^2 \cdot \left(1-\gamma^2\right). \\ Thus, w_2 &= \sqrt{\frac{2\alpha}{p\left(1-\gamma^2\right)}} \end{aligned} \tag{5}$$

Substituting these values in the throughput equation,

$$\lambda = \frac{\sqrt{\frac{\alpha(1+\gamma)}{2(1-\gamma)}}}{RTT\sqrt{p}} \tag{6}$$

It is evident from the above result that the throughput of the TCP-DCR protocol is similar to that of TCP Reno [23].

2.5 Network dynamics due to delayed congestion response

Even though TCP-DCR responds to a congestion signal with a delay of τ (one RTT) our results indicate that the response to congestion is faster than some of the other proposed protocols [22],[24] which are shown to be TCP-compatible. The earlier studies have shown that even in dynamic network conditions, the slowly responding protocols are fair and safe for deployment [25]. Since DCR responds to congestion faster than these earlier protocols, we expect DCR will be safe even in dynamic network conditions. We demonstrate this point through simulations, later in the paper.

3 TCP-DCR and Reordering Robustness

In current networks, packet reordering is observed to be not negligible [1],[2]. Many new design alternatives for routers or network architectures may benefit if there are no strict restrictions of zero packet reordering. When routers are designed based on parallel forwarding or when multi-path routing is employed, packet reordering could occur. In architectures such as diffserv, requiring no packet reordering restricts the choices for handling the multiple classes of packets of a single flow. As a result, packet reordering is becoming an even more important issue.

Several different solutions have been proposed in literature to solve this problem. In [4] the authors present several schemes which use DSACKs [26] or timestamps [27] are used to identify a *false fast retransmit*, where a packet is falsely identified by the sender as congestion loss and responded with a reduction in sending rate. In response to finding that a false fast retransmit has occurred, the sending rate is restored back to the level it was before the false fast retransmit. The reordering length for the packet is measured using the information available from DSACKs and the *dupthresh* is increased to avoid false fast retransmits. If a RTO timeout occurs, then it is presumed that the *dupthresh* has grown too large and it is reset to 3. In [3] this process is further refined at the cost of maintaining significantly more state at the sender and using complicated algorithms for finding the optimal value for *dupthresh* such that costly RTO timeouts are avoided, while the performance is optimized to provide maximum reordering robustness.

These schemes rely on some additional scheme for identifying reordering in the network (such as DSACKs or timestamps) and the perceived reordering information is collected from the network to set an optimal value for *dupthresh*. The intent is to estimate the optimal amount of time to delay the triggering of fast retransmit algorithm to provide maximum reordering robustness, without resorting to RTO timeouts too often.

By using TCP-DCR, this goal can be met without having to use complex state or algorithms for tuning the value of *dupthresh*. While TCP-DCR does not tune the *dupthresh* based on the perceived reordering in the network, when it is set to one RTT, it provides a simple and effective mechanism for providing reordering robustness without causing RTO timeouts.

3.1 Simulation Results

We evaluated the performance of TCP-DCR using the ns-2 simulator [20] (version 2.26). The network topology is as shown in Fig. 3. The *n* different sources are connected to the router R1 which in turn is connected to the router R2. The *n* different receivers are connected to the router R2. The link between the routers R1 and R2 is configured to be the bottleneck link in experiments simulating congestion in the network. The default values for the link bandwidth and delay for the links between the routers and the end nodes is fixed at 10 Mbps and 1 ms respectively. The link 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.

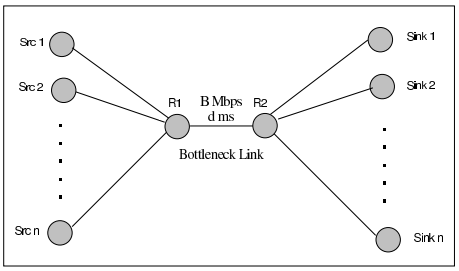

Fig. 3. Network Topology Used in ns-2 Simulations

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. This allows us the flexibility to choose the percentage of the packets to be delayed, the distributions for choosing the packets randomly as well as the distribution for the delays.

The TCP-DCR agent is implemented by modifying the tcp-sack1 implementation of TCP-SACK agent in ns-2. 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.

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 we show the effect of delayed packets on TCP-SACK, as the percentage of the packets getting delayed is increased, and the corresponding improvement in performance in case of TCP-DCR. The packet delay is picked from a normal distribution with a mean of 25ms and a standard deviation of 8ms. Thus, most packets chosen for delaying are delayed in the range 0 to 50ms, simulating mild but persistent reordering. The bottleneck link bandwidth is set to 8Mbps and the delay to 50ms. The receiver advertises a very large window such that the sending rate is not clamped by the receiver dynamics. 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. The X-axis shows the percentage of packets being delayed, and the Y-axis shows the total throughput in packets for the flow. Fig. 4 shows the results As can be seen from the graph, the performance of TCP-SACK degrades rapidly, since the reordered packets are treated as indication of congestion in the network and the sending rate is reduced. Persistent reordering makes the sender congestion window stay at a small value 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 within the delay in the congestion response τ. As a results, DCR does not reduce its window and its performance remains close to the performance of TCP-SACK with zero packets delayed in the network.

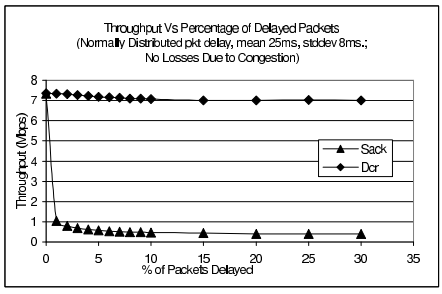

Fig. 4. Throughput Vs Percentage of Packets Delayed (With Single Flow)

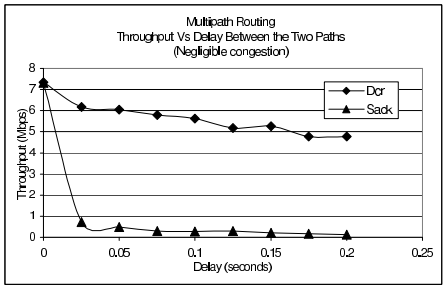

Fig. 5. Performance Comparison with Multi-path Routing

Performance Comparison with Multi-path Routing One of the situations in which more robustness to reordering is very important 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 by the difference between the round trip time of the two routes. 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.

Packets are alternately sent over this link and the link with the larger link delay. There is no congestion in the network. Fig. 5 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.

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. In order to study this, we conducted several simulations with multiple flows in the network, with half the flows using TCP-DCR and the other half of the flows using TCP-SACK. The graphs are plotted showing the average throughput achieved by each type of flow with an error bar showing the range of throughput for individual flows.

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. 6 shows the results when 10% of the packets are delayed.

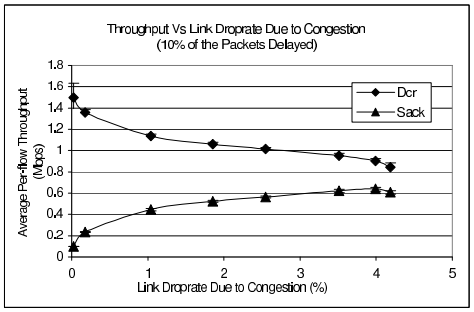

Fig. 6. Throughput Vs Link Droprate due to Congestion

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 fact that TCP-DCR realizes 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.

Mixed Workload at Varying Packet Delay Rates We now revisit the experiment showing the performance comparison of TCP-DCR and TCP-SACK at different packet delay rates, but with TCP-DCR and TCP-SACK flows vying for the same bottleneck link capacity. The experimental setup is similar to that explained above, with the only difference that the congestion in the network is fixed at 1.5%. Fig. 7 shows the results.

When there is no reordering in the network, the link capacity is shared equitably by the different flows. When more and more packets are delayed, the TCP-SACK flows suffer unnecessary reductions in sending rate and their average throughput reduces. The reduction in the average throughput of the TCP-SACK flows is not as drastic as seen in Fig. 4. When one SACK flow reduces its window due to a reordered packet, other SACK flows can claim some of this bandwidth and hence the differences in throughput become smaller with multiple flows. It is observed that TCP-DCR achieves 2-3 times more performance when network reorders packets. It is to be noted again that this gain is not due to DCR's unfairness or aggressiveness, but due to correctly recovering from packet reorder events (when SACK does not do so).

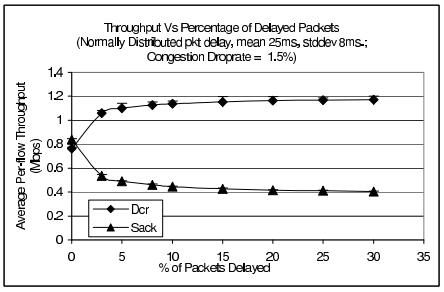

Fig. 7. Mixed Workload at Varying Packet Delay Rates

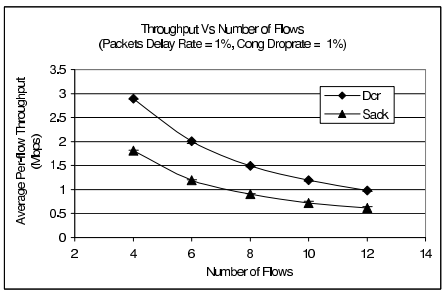

Fig. 8. Mixed Workload with Varying Number of Flows

Mixed Workload with Varying Number of Flows In this simulation, we study the throughput of individual flows as the number of flows in the workload is varied. The experimental setup is similar to that mentioned above. Congestion in the network is maintained at around 1% by adjusting the buffersize at router R1 for the link R1-R2, and 1% of the packets are delayed. The number of flows in the network is varied from 4 to 12, with half the flows using TCP-DCR and the other half of the flows using TCP-SACK. Fig. 8 shows the results.

As seen from the results, the difference between the average throughput of TCP-DCR flows and TCP-SACK flows remains fairly stable, indicating that the performance is consistent even when the number of flows in the network is varied.

Comparison with Other Protocols Our results can be directly compared with the results presented in [3] and other solutions proposed for handling reordering. The results show that DCR performs as well or better than the earlier solutions when packet reordering in the network is significant.

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 [5]. 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: the connection between the sender and receiver is split into two separate connections, one between the fixed sender and the base station and the other between the base station and the mobile receiver. The losses that are not related to congestion are recovered by the connection between the base station and the mobile host, and hence hidden from the fixed sender. [6],[7],[8],[9] (b) TCP-aware link layer protocols: the link layer is aware of the semantics of the TCP protocol and the dupacks are suppressed from reaching the sender if it can be recovered by link level retransmission. [10],[11] (c) Explicit loss notification approaches : TCP sender relies on the network to provide explicit notification about the error type [12],[13],[14]. (d) Receiver-based approaches : approaches where receivers either delay dupacks [16] or compute the desired sending rate [15]. (e) Modifications to TCP : some of the modified TCP algorithms have been shown to improve the performance of TCP [18],[19]. TCP-DCR modifies the sender side of TCP and relies on link level retransmission for recovering from channel errors. Earlier work has shown that local recovery of channel errors is efficient [17].

When both congestion losses and losses due to the transmission errors can occur, the 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. In order to maintain the segregation between the different layers of the TCP/IP stack, the link layer should not be required to know the semantics of the transport level protocol and the transport layer should not expect explicit notification about the type of the loss from the network layer. 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.

4.1 Simulation Results

The network topology used in these simulations is similar to the one shown in Fig. 3, 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 and delay is kept fixed at 1 Mbps and 20 ms respectively, unless otherwise mentioned.

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.

We conducted several different experiments, and the results show similar trends as indicated by the results for reordering robustness. Due to the lack of space and in the interest of avoiding repeatative results, we present only the important results here. Interested readers may find the other results in [28].

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. 9. The workload consists of a single flow in this case. There is no congestion in the network.

As can be seen from the graph, TCP-DCR performs significantly better than TCP-SACK. Since there is no congestion in the network, all the packet losses are due to channel errors. Due to delayed response algorithm, TCP-DCR postpones the window reduction upon receiving dupacks. This allows the link layer retransmission scheme time to recover the lost packets thereby making a window reduction unnecessary. Thus, when there is no congestion in the network, 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, due to repeated reductions of the congestion window, TCP-SACK cannot efficiently utilize the network bandwidth, especially so at high channel error rates.

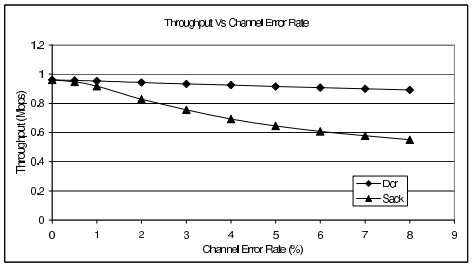

Fig. 9. Throughput Vs Channel Error Rate

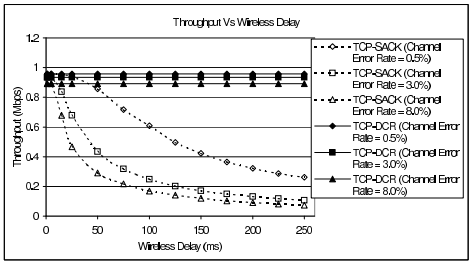

Fig. 10. Throughput Vs Wireless Link Delay

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[30, 31]. 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. 10 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 long time for the protocol to increase the window to the correct value again. This results in underutilization of the network bandwidth. TCP-DCR on the other hand is more robust in the face of large wireless delays, since the window is not reduced as often.

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 ate 1Mbps and 20ms. Fig. 11 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. This is not because the TCP-DCR flows are more aggressive than TCP-SACK. Rather, it is due to the fact that the TCP-DCR flows can make use of the link bandwidth *not utilized effectively* by the TCP-SACK flows. Recall from the discussion in previous sections that the TCP-SACK flows cannot utilize the available bandwidth completely at high channel errors because of persistent window reductions. The TCP-DCR flows claim this share of the bandwidth not used by the TCP-SACK flows. So when the congestion in the network is low, the TCP-DCR flows help improve the link utilization without starving the 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.

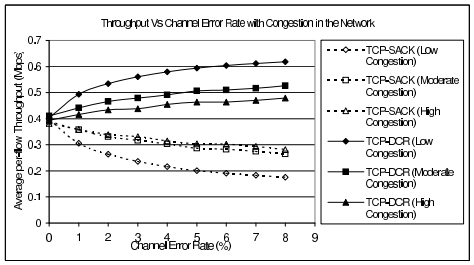

Fig. 11. Throughput Vs Channel Error Rate with Congestion in the Network

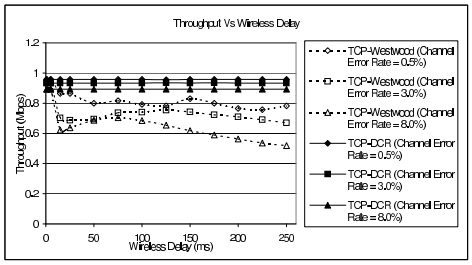

Fig. 12. Performance Comparison of TCP-DCR Vs TCP-Westwood

4.2 Comparison of TCP-DCR with other protocols

We have carried out extensive simulations to compare the performance of DCR with other protocols, particularly, TCP-Reno and TCP-Westwood. Due to lack of space, we have included only one of the results, showing the performance comparison of TCP-DCR with TCP-Westwood at different wireless delays and channel error rates in Fig. 12. The WestwoodNR agent was used in this simulation in the ns-2.26 version. The wireless link bandwidth is fixed at 1Mbps and the receiver advertised window and the wireless link buffer size are adjusted to maximize the link utilization even at large delays. Congestion losses occur only at the bottleneck link router. The simulations indicate that at low channel errors and low delays, the performance of both the protocols flavors are similar. At higher channel error rates and large delays, TCP-DCR performs better.

5 TCP-DCR with zero non-congestion events

The earlier two sections 3 and 4 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?

In this section, we study these questions at multiple levels. We try to understand the impact of congestion response delay on (a) the performance of individual flows (throughput, delays and fairness), (b) the TCP characteristics (RTT estimation, timeout calculation etc.) and (c) the network (average queue lengths, drop rates etc.).

To understand these issues, we carry out different simulations. In most cases we present the results for three different scenarios. In the first scenario, all the flows are TCP-DCR. In the second scenario, all the flows are TCP-SACK and in the third, 50% of the flows are TCP-DCR and 50% of the flows are TCP-SACK. We compare the flow characteristics, TCP characteristics and the network characteristics for these three scenarios to gauge the impact of delaying the congestion response when the network is free of non-congestion events. The topology used in these simulations is similar to that in section 3.

5.1 Fairness

The results in section 3.1 and section 4.1 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 3.1. The network consists of 12 flows with 6 of them using TCP-SACK and the other 6 of them use TCP-DCR. The congestion response delay is based on the ack-based implementation. The graph shows

the average throughput realized by DCR and SACK flows. From Fig. 13 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. In accordance with the analysis, these results indicate that TCP-DCR does not behave more aggressively than TCP-SACK, when τ is set to one RTT.

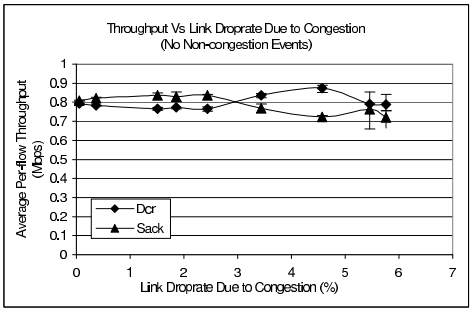

Fig. 13. Throughput comparison with Zero Non-congestion Events.

5.2 Packet Delivery Time

Since DCR delays the congestion response by one RTT, DCR 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. Results are presented for the three different scenarios explained earlier. The network buffersize is set to about 5 * DelayXBandwidth product (60 packets) and the average congestion droprate across the bottleneck link is 2.5%. The simulation is run for 200 seconds.

Fig. 14 shows the plot of packet delivery times for a randomly chosen DCR flow and a randomly chosen SACK flow against the packet sequence number. The plots show the packet delivery times are scattered in two regions. The dense population of points around 0.05 seconds represent the packets that are delivered normally. The points with larger delay represents packets delayed due to larger instantaneous queue lengths and the packets that are recovered through retransmission. In scenario 1 (100% DCR flows), the average packet delivery time for packets of the sample flow recovered via retransmission is 207ms. In scenario 2 (100% SACK flows), it is 178ms. In scenario 3 (50% DCR and 50% SACK), the average packet delivery time for retransmitted packets of the sample DCR flow is 201ms and for SACK, it is 182ms. 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.

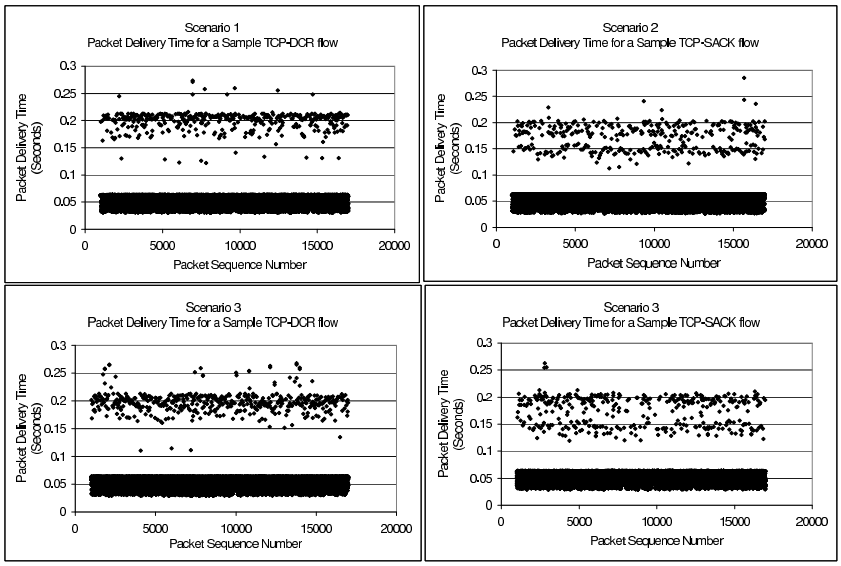

Fig. 14. Packet Delivery Time with Zero Non-congestion Events.

5.3 RTT Estimates

As explained in the above section, delaying the congestion response of TCP by one RTT can increase the packet recovery time of lost packets but the packet delivery time for the rest of the packets is similar to that in any standard implementation of TCP. 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. Fig. 15 shows the instantaneous RTT estimated by the TCP source as a function of time for the three scenarios mentioned above. These results agree with the discussion above.

5.4 Summary of other observations

We conducted several other experiments to evaluate the queue lengths, the timeouts, the perflow droprates, and the link utilizations to understand the impact of DCR flows on the network characteristics. We summarize these results in Table 16.

As seen from these results, DCR flows do not drastically alter the observed network characteristics compared to a network with only SACK flows.

5.5 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

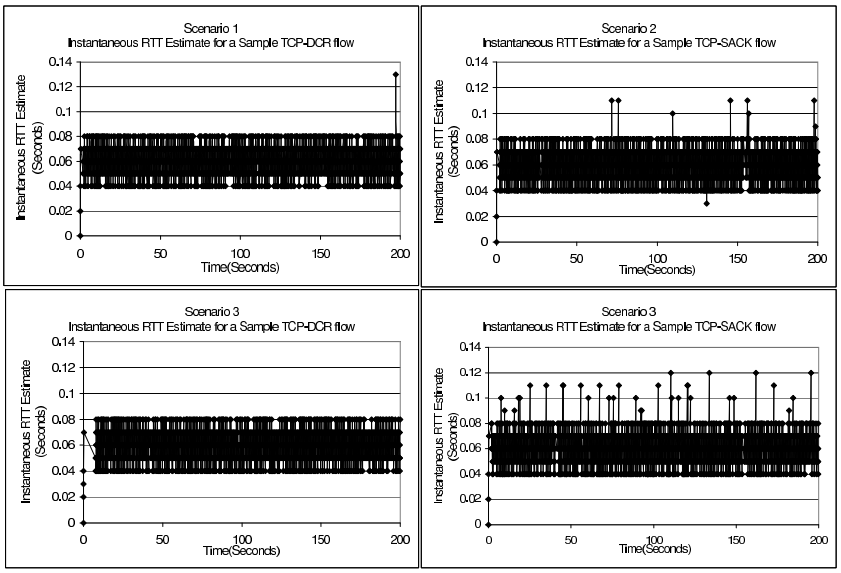

Fig. 15. Instantaneous RTT Estimates with Zero Non-congestion Events.

Scen-ario	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. 16. Summary of observations with Zero Non-congestion Events.

flows were added. We compared the response of TCP-DCR to that of TCP-SACK for this sudden increase in traffic. Fig. 17 shows the results.

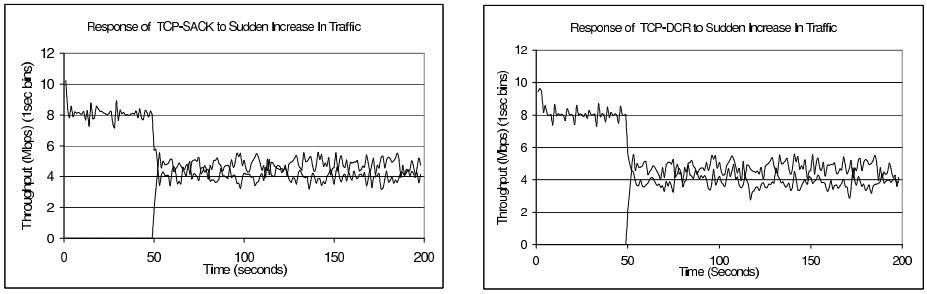

Fig. 17. Response to Sudden Increase in Traffic

It can be seen from the graph that the response of TCP-DCR is similar to that of TCP-SACK. Even though TCP-DCR delays the congestion response, the newly added TCP-SACK flows and the previously stabilized TCP-DCR flows start sharing the network bandwidth equitably within the same time as in the case of TCP-SACK. In order to quantify the reaction time to sudden changes in load, we computed the time it takes for existing flows to drop down to 55% of the link capacity, thus allowing the new flows to achieve 45% of the link capacity. The time to reach (55%, 45%) allocation for TCP-SACK was 3.1 seconds and for TCP-DCR, it was 3.67 seconds. This shows that TCP-DCR is not worse than TCP-SACK in responding to sudden increases in traffic load.

Interaction with Web-like Traffic In this section we evaluate the performance of TCP-DCR and TCP-SACK when competing with a traffic mix of several short-term flows simulating web-transfers. The network consists of 8 long-term ftp flows(TCP-SACK or TCP-DCR) and 500 web-like flows(TCP-SACK). The transfers are started at around 0 seconds with a staggering of 1ms to avoid synchronization. Each short-term flow sends N packets after T seconds from the start of its previous transfer. N is drawn from a uniform distribution between 10 and 20 and T is drawn from a pareto distribution with mean 15 seconds, simulating the different request sizes and user think-times. The random variable generators for the short-term flows are seeded with the flow id, so that any given flow has a fixed pseudo random sequence. This ensures that when the simulation is first run with TCP-SACK ftp transfers and then repeated with TCP-DCR ftp transfers, the random variables used in simulating the web transfers, have the same value. The average link droprate over the period of the simulation is 3%. Fig. 18 shows the aggregate throughput of the long term flows and the traffic (computed with 1 second bins) against time.

In the case of TCP-SACK, the aggregate throughput of TCP-SACK flows over the sim ulation period is 4.76Mbps, and that for the web traffic is 4.84Mbps. The aggreg ate throughput of TCP-DCR flows is 4.73Mbps and that for the web traffic is 4.82 Mbps.

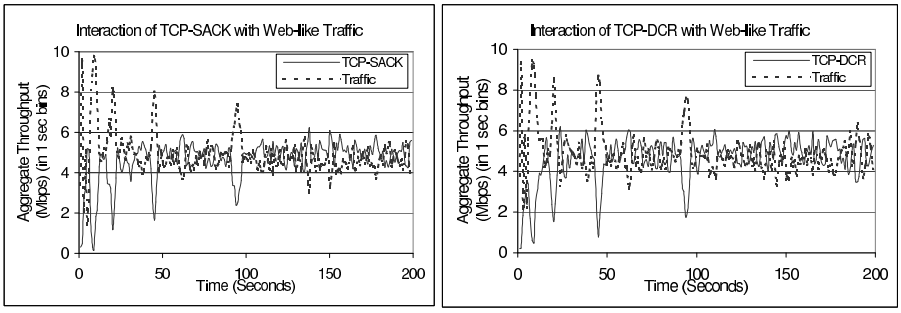

Fig. 18. Interaction with Web-like Traffic

This indicates that the interaction of the TCP-DCR flows with short-term w eb traffic is similar to that of TCP-SACK.

6 The Linux Implementation of TCP-DCR

In order to test the behavior of TCP-DCR on a real network, we implemented the TCP-DCR modification on the Linux 2.4.20 kernel. The evaluation is currently in progress and in this section we provide a brief overview of the preliminary results. The modifications are simple. The threshold on the number of dupacks required for triggering fast retransmit/recovery algorithm is modified, to the current window size, upon the receipt of the first dupack.

The experimental setup is shown in Fig. 19. All three machines are Linux hosts. Only the Linux code at the source is modified. The receiver and the router use the default networking code available in the Linux 2.4.20 kernel. Additionally, the router runs the network emulation package called NIST Net [32] to allow us to control the link delay and delay variance. By modifying the delay, we can emulate different types of networks such as LAN/WAN and by modifying the delay variance, we can cause packet reordering. All the NICS used operate at 10Mbps and the MTU is 1500 bytes. The default *pfifo* queue discipline is used at the router with the queue length of 100 packets.

We present here the preliminary result for LAN-like delays, where the delay in the NIST Net emulator is fixed at 10ms. The delay variance at the router is modified to cause different levels of packet reordering. WEB100 module [33] was used at the sender to collect detailed statistics regarding the TCP behavior. TCPDump[34] and TCP Trace [35] were used for collecting the statistics at the receiver. Table 20 shows the results for the FTP transfer of a 33MB file between the sender and the receiver.

The results show that the throughput of TCP-DCR remains fairly constant as the percentage of out of order packets in the network is increased. Similar results were observed in a WAN-like setting where the NIST net emulater delayed packets by 100ms. We are currently conducting additional experiments to evaluate the performance of TCP-DCR further.

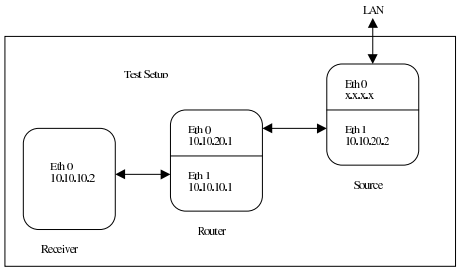

Fig. 19. Experimental setup for evaluating TCP-DCR modifications to Linux

Delay Variance (ms)	RTT (ms) (min/avg/max/mdev)	% OOO packets	Throughput (Kbps)
0	16.308/16.444/16.585/0.126	0	1127.29
1	15.241/16.520/19.278/0.910	13.90	1106.52
2	14.223/16.330/22.674/1.786	30.38	1087.45
3	12.781/16.247/23.585/2.322	39.87	1093.90

Fig. 20. Performance of TCP-DCR on Linux Implementation with Packet Reordering

7 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 have provided an analysis to show that DCR achieves similar throughput as regular TCP in steady state conditions. We studied DCR's handling of non-congestion events in two specific cases, namely, packet reordering and wireless channel errors. In both cases, 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. 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 presented the preliminary results here. We plan to test it further more rigorously.

8 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. 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.
6. A. Bakre and B. R. Badrinath, "I-TCP: indirect TCP for mobile hosts," *Proceedings of the 15th. International Conference on Distributed Computing Systems (ICDCS)*, May 1995.
7. R. Yavatkar and N. Bhagawat, "Improving End-to-End Performance of TCP over Mobile Internetworks," *Workshop on Mobile Computing Systems and Applications*, December 1994.
8. K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *ACM Computer Communications Review, vol. 27, no. 5*, 1997.
9. K.-Y. Wang and S. K. Tripathi, "Mobile-end transport protocol: An alternative to TCP/IP over wireless links," *IEEE INFOCOM'98, vol. 3, p. 1046*, 1998.
10. H. Balakrishnan, S. Seshan, E. Amir and R. Katz, "Improving TCP/IP performance over wireless networks," *Proc. of ACM MOBICOM*, Nov. 1995.
11. H. M. Chaskar, T. V. Lakshman, and U. Madhow, "TCP Over Wireless with Link Level Error Control: Analysis and Design Methodology", *IEEE Trans. on Networking, vol. 7, no. 5*, Oct. 1999.
12. H. Balakrishnan and R. H. Katz, "Explicit Loss Notification and Wireless Web Performance," *Proc. of IEEE GLOBECOM*, Nov. 1998.
13. K. Ramakrishnan and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP," *RFC 2481*, January 1999.
14. R. Krishnan, M. Allman, C. Partridge and J. P.G. Sterbenz, "Explicit Transport Error Notification for Error-Prone Wireless and Satellite Networks," *BBN Technical Report No. 8333, BBN Technologies*, February, 2002
15. P. Sinha, N. Venkitaraman, R. Sivakumar and V. Bhargavan, "WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks," *Proceedings of ACM MOBICOM*, August 1999.
16. 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.
17. D. Eckhardt and P. Steenkiste, "Improving Wireless LAN Performance via Adaptive Local Error Control," *Proceedings of IEEE ICNP*, Austin, TX, 1998.
18. M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, "TCP selective acknowledgment options," *Internet RFC 2018*.
19. 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.
20. ns-2 Network Simulator. **http://www.isi.edu/nsnam/**
21. M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit," *RFC 3042, Proposed Standard*, January 2001.

22. D. Bansal and H. Balakrishnan, “Binomial Congestion Control Algorithms,” *Proceedings IEEE INFOCOM*, 2001.
23. J.Padhye, V.Firoiu, D.Towsley, and J.Kurose, “Modeling TCP throughput: A simple Model and its empirical validation,” *ACM SIGCOMM ’98*, Oct. 1998.
24. S. Floyd, M. Handley, J. Padhye, and J. Widmer. “Equation-based congestion control for unicast applications,” *Proc. of ACM SIGCOMM*, 2000.
25. D. Bansal, H. Balakrishnan, S. Floyd and Scott Shenker, “Dynamic Behavior of Slowly Responsive Congestion Control Algorithms,” *Proceedings of ACM SIGCOMM*, Sep. 2001.
26. Sally Floyd, Jamshid Mahdavi, Matt Mathis and Matt Podolsky, “An Extension to the Selective Acknowledgement (SACK) Option for TCP,” *RFC 2883*, July 2000.
27. R. Ludwig and M. Meyer, “The Eifel Detection Algorithm for TCP,” *RFC 3522*, April 2003.
28. 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.
29. M. Allman, V. Paxson and W. Stevens, “TCP Congestion Control,” *RFC 2581*, April 1999.
30. M. Allman, D. Glover and L. Sanchez, “Enhancing TCP Over Satellite Channels using Standard Mechanisms,” *RFC 2488*, January 1999.
31. J. Border, M. Kojo, J. Griner, G. Montenegro and Z. Shelby, “Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations,” *RFC 3135*, June 2001.
32. The NIST Net network Emulator. **http://snad.ncsl.nist.gov/itg/nistnet/**
33. The Web100 Project. **http://www.web100.org/**
34. The *tcpdump* utility. **http://www.tcpdump.org/**
35. The *tcptrace* tool. **http://www.tcptrace.org/**