TCP-DCR: A Novel Protocol for Tolerating Wireless Channel Errors

Sumitha Bhandarkar, *Student Member*, *IEEE Computer Society*, Nauzad Erach Sadry, A.L. Narasimha Reddy, *Senior Member*, *IEEE Computer Society*, and Nitin H. Vaidya, *Senior Member*, *IEEE Computer Society*

Abstract—This paper presents TCP-DCR, a set of simple modifications to the TCP protocol to improve its robustness to channel errors in wireless networks. TCP-DCR is based on the simple idea of allowing the link-level mechanism to recover the packets lost, due to channel errors, thereby limiting the response of the transport protocol to mostly congestion losses. This is done by delaying the triggering of congestion response algorithms for a small bounded period of time τ to allow the link-level retransmissions to recover the loss due to channel errors. If at the end of the delay τ the packet is not recovered, then it is treated as a packet lost due to congestion. We analyze TCP-DCR to show that the delay in congestion response does not impact the fairness towards the native implementations of TCP that respond to congestion immediately after receiving three dupacks. We evaluate TCP-DCR through simulations to show that it offers significantly better performance when channel errors contribute more towards packet losses in the network with no or minimal impact on the performance when congestion is the primary cause for packet loss. We also present an analysis to show that protocol evaluation in the wireless networks is significantly influenced by the number of flows in the network.

Index Terms—Wireless network, channel errors, TCP, delayed congestion response, local recovery.

1 Introduction

The popularity of wireless networking has increased dramatically over the past few years. However, integrating high delay, high channel error wireless networks with existing wired networks still poses significant challenges to the research community. Incorporating end-to-end congestion control for wireless networks is one of the primary concerns. Considering that TCP is the most prevalent congestion control protocol used on the wired Internet, compatibility issues would necessitate its use on wireless networks as well. But, TCP was designed to work well in networks with low channel error rates and when used in wireless networks which are generally characterized by larger channel error rates, the losses due to channel errors also get treated as congestion losses, resulting in suboptimal performance [1].

When both congestion losses and losses due to the transmission errors can occur, a simple solution would be to let the link-layer mechanisms recover the losses due to transmission errors and the transport protocol to recover the losses due to congestion. 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. Based on these ideas and the additional requirement that the solution should be simple and incrementally deployable, in this paper, we propose *Delayed Congestion Response* TCP protocol (TCP-DCR).

TCP-DCR works in conjunction with a simple link-level protocol to provide the benefits of standard TCP implementations without the associated degradation in performance due to channel errors in wireless networks. In TCP-DCR, the response to the receipt of duplicate acknowledgements (henceforth referred as dupacks) is delayed by a short bounded period τ. If the packet is recovered by link-level retransmission before the end of the delay period τ (indicated by the receipt of a cumulative acknowledgement acknowledging the lost packet), TCP-DCR proceeds as if the packet loss never occurred. However, if the packet is not recovered by link-level retransmission by the end of the delay period, TCP-DCR protocol triggers the congestion recovery algorithms of fast retransmission and recovery. By doing this, we effectively change the paradigm of TCP that *all losses are due to congestion* to the paradigm that *all losses are due to channel errors for a period of* τ. It may be noted here that no changes need to be made for the TCP at the receiver and base stations are not required to maintain any TCP-related state information.

The rest of the paper is organized as follows: In Section 2, we take a brief look at some of the existing solutions to improve performance of TCP over wireless networks. In Section 3, we provide the detailed analysis and discussion of TCP-DCR. In Section 4, we present an evaluation of the TCP-DCR protocol using simulations. Section 5 wraps up the paper by taking a look at the conclusions and open issues regarding TCP-DCR.

- *S. Bhandarkar and A.L.N. Reddy are with the Department of Electrical Engineering, Texas A & M University, College Station, TX 77843-3128. E-mail: {sumitha, reddy}@ee.tamu.edu.*
- *N.E. Sadry is with Z-Force Incorporated, Laguna Hills, CA 92637. E-mail: nauzads@cs.tamu.edu.*
- *N.H. Vaidya is with the Department of Electrical and Computer Engineering, and Coordinated Science Laboratory, University of Illinois, Urbana-Champaign, IL 61801. E-mail: nhv@uiuc.edu.*

Manuscript received 20 Feb. 2003; revised 28 Oct. 2003; accepted 16 Feb. 2004; published online 27 July 2005.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0021-0203.

2 Related Work

Over the past few years, several solutions have been proposed to improve the performance of TCP over wireless networks. These solutions fall in one of the following broad categories: split connection approaches, link-layer schemes, explicit loss notification approaches, receiver-based approaches, and modifications to TCP. In 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. I-TCP [2], MTCP [3], M-TCP [4], and METP [5] are examples of this approach. Some of these approaches do not maintain the end-to-end semantics of TCP. These protocols may require state to be maintained and packets to be buffered at the base station.

In the link-layer schemes, the losses due to transmission error are recovered locally by the link layer. Such schemes can be purely local such as [8] or aware of the semantics of the TCP protocol such as [6], [7].

The explicit loss notification approaches like ELN [9], ECN [10] and ETEN [11] provide the TCP sender with explicit notification that a loss has occurred. The sender can then decouple congestion control from retransmission to recover the packets lost, based on the type of notification. These schemes require that the receivers/network routers be able to distinguish the channel errors from congestion losses and be capable of marking the acknowledgements with appropriate notification. The senders then respond to the notification. Such approaches require modifications to network infrastructure, the receivers, and the senders.

WTCP for WWANs [12] is a receiver-based approach where receiver computes the desired sending rate using rate control algorithm and notifies the sender of this rate using the acknowledgements. The receiver has to do considerable processing to compute the statistical information regarding losses and observed RTT. Another receiver-based approach is the Delayed Dupack scheme [13] which closely imitates the snoop protocol [6] at the receiver, so that the link-level scheme need not be TCP-aware. In this scheme, the third and subsequent dupacks are delayed for a bounded period of time to allow the link-layer time to recover the packet. If the packet is recovered within the delay period, the dupacks are not sent, otherwise, all the dupacks are released.

It has also been shown that, by using TCP-SACK [14] or TCP-Westwood [15] instead of TCP Reno, performance can be improved. However, the performance improvement gained by using TCP-SACK protocol, is due to its ability to recover from multiple losses in one RTT and does not necessarily indicate robustness to channel errors. TCP-Westwood (referred henceforth as TCPW) aims at distinguishing the losses due to congestion in the network from other random losses. In TCPW, a rate estimator that estimates the fair rate by sampling and exponentially filtering the acknowledgements dictates the window reduction. TCPW algorithm has been shown to perform better than TCP Reno when the transmission loss rate is large. In this paper, we advocate the use of TCP-DCR modifications with the TCP-SACK flavor. The simplicity of this approach, in our opinion, makes it a far more compelling solution than other TCP-based solutions or non-TCP-based solutions for improving the robustness of TCP to channel errors in the wireless networks.

3 Discussion

In this section, we provide a detailed description for the Delayed Congestion Response TCP (TCP-DCR) modifications. Traditional implementations of TCP assume that packet losses are primarily due to congestion in the network. As a result, when a packet loss is indicated either by the receipt of three DUPACKs or a time-out of the retransmission timer, it embarks on congestion control and packet recovery. This may not be appropriate in a wireless network, where a significant amount of the losses in the network could be due to channel errors. The TCP-DCR modifications aim to remedy this by changing *the time at which* the fast retransmit/recovery algorithms are triggered. The receipt of dupacks is assumed to be caused by channel errors, for a bounded delay period τ. If the packet loss is indeed due to channel errors and the link layer supports local recovery, then the packet is recovered by the link-level retransmission, and our presumption is correct. However, if by the end of the delay period, the packet is still not recovered, the presumption that the packet loss is due to channel errors is abandoned and the packet is recovered using the fast retransmission and recovery algorithms.

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. In this section, we look at the behavior of TCP-DCR under different types of losses, the choice of τ, the implementation details, assumptions about the link-level retransmission scheme, and the analysis of the steady state bandwidth for TCP-DCR.

3.1 Behavior of TCP-DCR

Fig. 1 shows the graphical representation of TCP-DCR when a) the loss of a packet is due to transmission errors and b) the loss of a packet is due to congestion. The TCP-DCR sender sends packets 1 through 5. However, due to channel error, say, packet 2 is lost. This is communicated by the link layer to the base station, say, by a negative acknowledgement (NACK). The base station immediately retransmits packet 2. But, before packet 2 is recovered by link-level retransmission, the TCP receiver sends dupacks for packet 2. In the case of the traditional implementations of TCP, three dupacks would trigger an immediate retransmission of packet 2 at the TCP sender, followed by an unnecessary window reduction. However, in the case of TCP-DCR, a delayed response timer of one RTT is started at the sender when the first dupack is received. During this delay period, packet 2 is recovered via link-level retransmission causing the TCP receiver to generate a cumulative acknowledgement acknowledging packet 2. On the receipt of this acknowledgement, the TCP-DCR sender cancels the delayed response timer, and the unnecessary retransmission of packet 2 and reduction in congestion window is avoided. Also, TCP-DCR sends one new packet on the

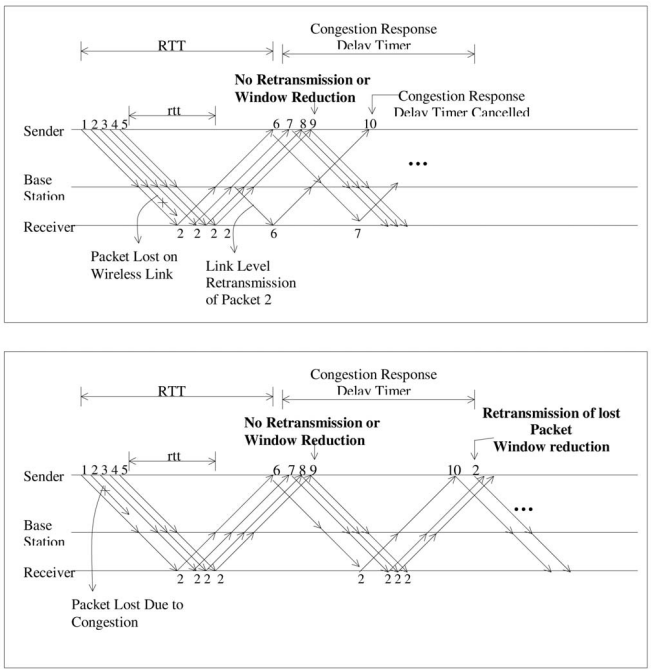

Fig. 1. Behavior of TCP-DCR.

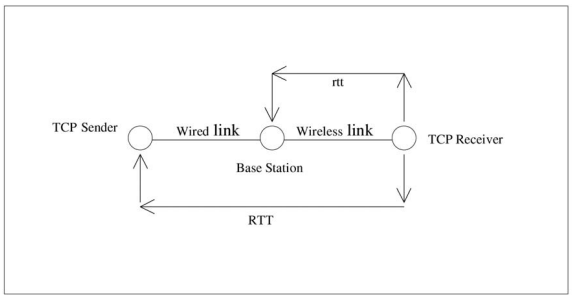

Fig. 2. Analysis of TCP-DCR with no congestion losses.

receipt of each dupack, if allowed by the congestion window, similar to the proposed standard "Limited Transmit Algorithm" [25]. This ensures that, during the delay τ, the sending rate of the TCP-DCR is the same as it was when the first dupack arrived.

In the case of a congestion loss, the packet cannot be recovered through link-level retransmission. Upon the receipt of the first dupack, the delayed response timer is started. However, since the packet is dropped by an intermediate router due to congestion, a cumulative acknowledgement for the lost packet is not received. When the timer expires, packet 2 is retransmitted and the congestion window is reduced to half. An important fact to remember here is that the delay of τ does not cause the TCP-DCR sender to dramatically oversend packets because the protocol is still ACK-clocked. That is, a new packet is sent only upon the receipt of a dupack and the sending rate during the delay period is atmost the sending rate when the first dupack arrived.

3.2 Choice of τ

It is clear from the discussion above that the choice of the delay τ determines the performance of TCP-DCR. 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, choosing the correct value for the delay is important. It is essentially a trade off between unnecessarily inferring congestion and unnecessarily waiting for a long time before retransmitting a lost packet. In this section, we provide guidelines for choosing reasonable bounds on the delay to make it useful, without adversely modifying the TCP behavior. Note that the current practice of waiting for three dupacks at the sender is merely a heuristic.

Fig. 2 shows a general scenario 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., interpacket 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 interpacket delays are nonzero 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.

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 milliseconds. 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.

Based on the discussion above, we conclude that a choice of waiting for one RTT after the first dupack before responding to congestion is reasonable. 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.

3.3 Implementation Details

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 [25]. 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 straightforward, 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 end up resulting 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 or channel errors) 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 to 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 [24] that the dupacks (or SACKs) are not delayed.

3.4 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 allows 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 losses due to channel errors.

3.5 Link-Level Retransmission Scheme

The performance benefits to be gained from using the TCP-DCR modifications depend heavily on the existence of an underlying scheme for recovering the losses due to channel errors. In this paper, 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 [22] 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.

A link-layer protocol that does not attempt in-order delivery in combination with the TCP-DCR protocol is well suited for satellite connections which are characterized by large round-trip delays. The wireless link continues to transmit subsequent packets while it waits for the ACK/NACK for a particular packet, thereby keeping the pipe full. If the packet is lost due to channel errors, then it is retransmitted and recovered at the link level without unnecessary reduction in sending rate at the transport level.

3.6 Analysis of Steady State Bandwidth

In this section, we present an analysis of the steady state bandwidth of TCP-DCR. The analysis is conducted along the similar lines of that presented in [19], [20]. This is an

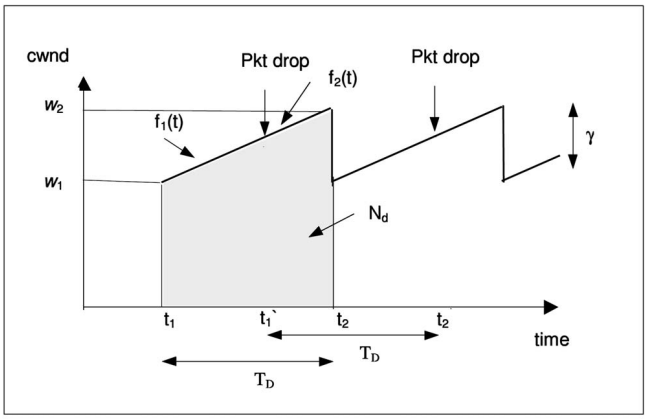

Fig. 3. Analysis of TCP-DCR with no channel errors.

approximate model aimed at capturing the behavior of TCP-DCR in networks with mild congestion, such that the protocol is mostly in the congestion avoidance state. When the TCP-DCR modifications are applied to the TCP-SACK variant, timeouts are largely avoided and this assumption is close to the real behavior of the protocol. The time between two successive packet losses is assumed to be deterministic. Under these assumptions, the congestion window behavior of TCP is cyclical and easier to analyze. This simplified model for analyzing TCP-DCR provides us with the relationship between the throughput and the packet loss rate and allows us to compare the same with a standard implementation of TCP under similar assumptions.

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 just as in traditional flavors of TCP. 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:

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

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. 3 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 (1), using continuous fluid approximation and linear interpolation of the window between w_t and w_{t+RTT}, we get

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

As can be seen from Fig. 3, 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 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 Fig. 3, 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,

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

N_D is the shaded area under the curve in Fig. 3. Hence,

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

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

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

Substituting these values in the throughput equation,

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

It is evident from the above result that the throughput of the TCP-DCR protocol is similar to that of a standard implementation of TCP that responds to congestion signals immediately after the receipt of three dupacks [19] and is not affected by the choice of τ. Even though, the TCP-DCR protocol continues to increase the congestion window and seems more aggressive than TCP during the delay period τ, the window reduction at the end of $RTT + \tau$ results in a larger decrease than the reduction at the end of RTT. So, the overall characteristics of the protocol are similar to that of TCP. However, in practice, increasing τ arbitrarily is not a recommended action, as this would delay relieving the congestion in the network. Moreover, it would delay the recovery of the lost packet by the TCP-DCR sender. Our analysis only applies to the case where a single packet is lost in a congestion window and an arbitrarily large τ would negate that assumption.

Based on the analysis in Section 3.2, we hypothesize that setting the delay in congestion response to one RTT would be an appropriate choice. This would allow sufficient time for the base station to recover the lost packet at the link layer, while relieving the congestion quickly.

3.7 Sender-Based Delay versus Receiver-Based Delay

Postponing the decision that the dupacks are caused by a packet loss due to congestion can be done at either the sender or the receiver. However, in the receiver-side transport layer scheme such as [13], it is difficult to find an optimal value for the delay since the receiver is unaware of sender's RTT estimates.

Also, when the delay is implemented at the receiver, be it at the transport layer or at the link layer [8], the ack-clock at the sender is lost. As a result, during the delay while the losses due to channel error are recovered, the sender does not send any packets and the flow remains idle.

In the case of small hand-held receivers, it may not be feasible to perform complicated processing at the receiver. In order to keep the receiver simple, it may be desirable to leave the processing to the sender. In addition, if the architecture is a client-server, by modifying one server, all the clients could benefit from improved performance.

Traditionally, in the design of TCP algorithms, most of the intelligence of flow and congestion control has been at the sender. It would be in tune with this practice to include the modifications at the sender.

4 Simulation Results

In this section, we present the evaluation of the TCP-DCR protocol based on simulations on the ns-2 simulator [21]. The TCP-DCR agent is implemented by modifying the TCP-Sack1 agent in ns-2. Timer-based delay is used for delaying the triggering of the fast retransmit/recovery algorithms. The TCP clock resolution is set to 10 ms (similar to Linux TCP). Upon receiving the first dupack, the congestion response delay timer is set. If a cumulative acknowledgement is received acknowledging the packet perceived to be lost, then the timer is reset. If the timer expires and the fast retransmit/recovery algorithms are triggered then, any additional "holes" are treated in exactly the same way as TCP-SACK would, irrespective of whether the holes are due to channel errors or congestion losses. The TCPSink agent is used for the receivers. The buffersize available at the receivers (indicated by the receiver advertised window) is set to at least twice the highest possible congestion window, to ensure the maximum performance improvement during the delay τ. Link-level retransmission is simulated by modifying the error model and the queue objects 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 rtt 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. FTP sources are used to generate traffic, which start sending data at time 0. In experiments where the topology consists of several flows, the start time of the different sources are staggered by 1 second to avoid synchronization. All simulations are run for 1,100 seconds, but data is collected only after the first 100 seconds to ensure that steady state is reached.

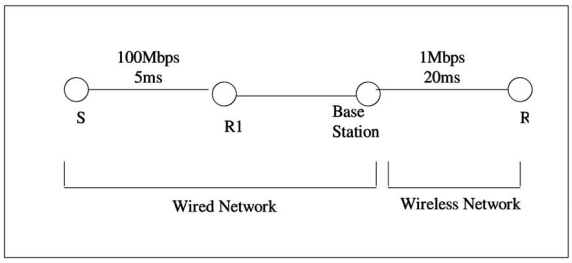

Fig. 4. Network topology for experiments with no congestion losses.

In these simulations, we compare the performance of TCP-DCR with the performance of TCP-SACK. Since TCP-DCR is the TCP-SACK protocol with the delayed triggering of the fast retransmit/recovery algorithm, the results give us an idea of the extent of performance improvements to be gained by simply delaying the congestion response by one RTT. It has been shown by earlier work [18], [16] that the impact of a slowly responding protocol on fairness, goodput, droprates, etc., are better when the bottleneck link router uses an active queue management scheme like RED. Since the aim is to find if, and by how much the behavior of TCP-DCR impacts unmodified TCP-SACK or the network, we have chosen to use Droptail queue management in our experiments. To be complete, we have carried out some experiments using RED as well and some of these results are reported in Section 4.2.6 and Section 4.2.7.

The results of the simulations are presented in three separate categories:

1. **Experiments with no congestion losses**. This category of simulations helps us understand the effect of channel errors on the performance of the protocols with and without the delayed congestion response.
2. **Experiments with only congestion losses**. It is important to evaluate how the TCP-DCR behavior differs from the behavior of the TCP-SACK protocol in the presence of congestion losses. In order to avoid interference from channel errors, in this category, we present results of simulations where the network has only congestion losses.
3. **Other Experiments**. This category presents results for scenarios where the network has both channel errors and congestion losses for low-delay wireless links as well as high-delay satellite links. In this category, we also present the results of the comparison with the TCP-Westwood [15] protocol.

By evaluating the TCP-DCR protocol in different scenarios, we aim to provide a comprehensive understanding of the protocol behavior with delayed congestion response.

4.1 Experiments with No Congestion Losses

The simple network topology shown in Fig. 4 is used for these experiments. The source S is connected to the router R1 which in turn is connected to the base station by wired links. The receiver R is connected to the base station by wireless links. The wired link bandwidth and delay is fixed at

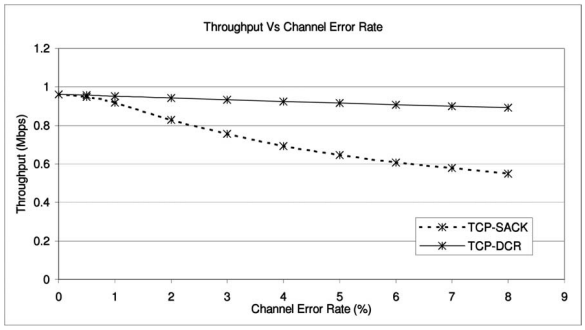

Fig. 5. Throughput versus channel error rate.

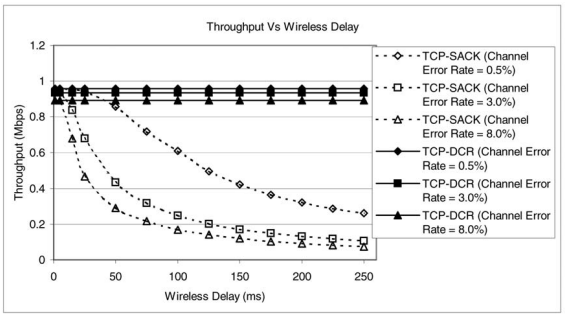

Fig. 6. Throughput versus wireless link delay.

100 Mbps and 5 milliseconds, respectively, and the buffersize is set to the delay-bandwidth product. The wireless link bandwidth, delay, and the buffersize are as shown in each individual simulation. The source S performs a single bulk data transfer to the receiver R with a packet size of 1,000 bytes.

4.1.1 Performance Comparison at Different Channel Error Rates

For this experiment, the wireless link bandwidth and delay are fixed at 1 Mbps and 20 milliseconds. The wireless link bandwidth is much smaller than the wired link bandwidth. In order to ensure no congestion losses occur, the receiver advertised window is fixed at 40 packets and the wireless link buffersize is fixed at 50 packets. In Fig. 5, the X-axis shows the channel error rates in percentage of the packets corrupted and the Y-axis shows the throughput in Mbps.

The experimental topology has been chosen to allow the TCP flows to maximize the link utilization without causing any drops due to congestion. However, in the case of TCP-SACK, since the source responds to the channel errors by reducing the sending rate, the throughput starts to deteriorate as the channel error rate increases. Even though the link capacity is small and the delay is relatively short, resulting is a relatively small rtt, the TCP-SACK flow cannot fully utilize the link. On the other hand, due to delayed congestion response algorithm, TCP-DCR postpones the window reduction upon loss notification. 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 is better than that of TCP-SACK and even at high channel error rates, it is comparable to the performance when there are no channel errors at all.

4.1.2 Performance Comparison at Different Wireless Delays

Some of the wireless networks, such as local wireless LANs, have delays of the order of a few milliseconds to a few with tens of milliseconds while the satellite links are characterized by much larger delays in the order of hundreds of milliseconds [26], [27]. In this section, we show the effect of the wireless delay on the performance of the different protocols. The wireless link bandwidth is fixed at 1 Mbps and the receiver advertised window and the wireless link buffersize are adjusted to maximize the link utilization even at large delays, without incurring congestion losses (125 packets and 150 packets, respectively). Fig. 6 shows the results. Throughput is plotted on the y-axis. The x-axis shows the different wireless delays.

It can be seen from the graph that the performance of TCP-DCR does not vary much when the wireless delay is varied. The performance of TCP-SACK, on the other hand, deteriorates drastically as wireless delays are increased. This is because at larger wireless delays, when the window is reduced, it takes a long time for the protocol to increase it back to the optimal value. This results in fairly degraded performance at higher link delays. TCP-DCR is more robust in the face of large wireless delays even at high channel loss rates.

4.1.3 Performance Comparison at Different Wireless Bandwidths

Improvement in wireless technology has been constantly raising the bar on how much bandwidth the wireless channels offer. We evaluate the impact of channel bandwidth on protocol performance. The wireless link delay is fixed at 20 milliseconds. The buffer size and the receiver window are adjusted for each simulation to allow maximum link utilization, without causing any congestion. Fig. 7 shows the results.

It can be seen from the graph that the TCP-SACK flows cannot utilize the link bandwidth well. At higher channel errors, due to persistent reduction in the sending rate the congestion window remains small, and no matter how much network bandwidth is available, the throughput of the TCP-SACK flow stays almost constant at a small value. TCP-DCR, on the the other hand, avoids reducing the congestion window for channel errors and, hence, is capable of utilizing the available bandwidth much more efficiently.

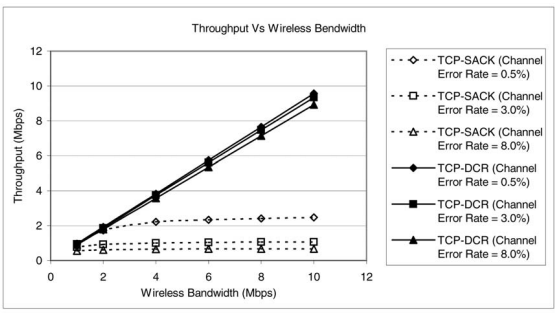

Fig. 7. Throughput versus wireless bandwidth.

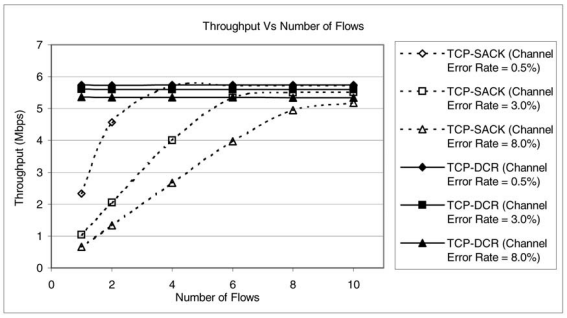

Fig. 8. Throughput versus number of flows.

4.1.4 Performance Comparison with Varying Number of Flows

At this point, we take a slight deviation to inspect an important factor to be considered while evaluating the new flavors of TCP protocol—the effect of the number of flows on the simulation results. An important observation made during the above experiments was that TCP-SACK flow was not able to completely utilize the bandwidth at high channel error rates and high wireless delays. It would seem intuitive then that, as the number of flows in the network is increased, the utilization of the link could be improved, because when one flow backs off in response to a packet loss, some other flow could utilize the link. So, we conducted a simulation where the wireless link bandwidth and delay were fixed at 6 Mbps and 20 milliseconds, but the number of flows between the source S and the receiver R was increased. The receiver advertised window and the buffersize are adjusted so that a single flow without any losses can almost fully utilize the link. However, note that, when the number of flows is increased, the congestion losses no longer remain zero. The results are presented in Fig. 8.

As expected, the link utilization does improve at higher number of flows. We have included these results in this paper to demonstrate an important point: Results for new protocols shown for just a fixed number of flows are not sufficient. In this case, for the network topology that we have chosen, by having a fixed number of flows greater than 8, TCP-SACK could be shown to provide very good performance even at very high channel error rates.

Another perspective on this issue can be provided by the following argument. It has been shown in [19] that the throughput of the TCP protocol is proportional to $\frac{1}{RTT*\sqrt{p}}$ (when timeouts are ignored), where p is the loss rate seen by a TCP flow and RTT is the round-trip time perceived by the TCP sender. When there is no congestion in the network, p represents only channel errors for TCP-SACK. These losses do not depend on the number of flows in the network and are fixed relative to the number of flows in the network. Then for any particular value of p and RTT, the throughput obtained by a TCP source is fixed, say at T. The fair share of bandwidth for any particular flow when there are n different flows in the network is B/n. When the value of n is chosen such that $B/n \leq T$, it will appear as if the protocol is making the best utilization of the available bandwidth, irrespective of how the protocol treats the channel errors.

Consider TCP-DCR on the other hand. As shown in the (6), the throughout of a TCP-DCR flow is also proportional to $\frac{1}{RTT*\sqrt{p}}$. However, in this case, p primarily represents the loss rate due to congestion in the network. As a result, when congestion in the network is zero, the throughout is only controlled by the receiver's window. In other words, when there is no congestion in the network, TCP-DCR can effectively utilize all the available bandwidth, irrespective of the number of flows in the network.

It might be tempting at this point to suggest that all we need, to improve the performance of TCP on a wireless network, is to fill up the pipe with many flows such that all the bandwidth can be utilized. This could probably be a feasible solution if we can ensure that at all times, there will be enough flows in the network to keep it fully utilized. However, if that is not the case, and we wish to have maximum utilization irrespective of how many flows are in the network, then we would require modifications to existing TCP protocols. Also, wireless technology is improving at a rapid rate, and as new technology becomes available, the bandwidth keeps increasing. The higher bandwidth would require a larger number of flows to keep the link fully utilized for the same channel error rate. It would be unreasonable to depend only on the number of flows in the network to make the best use of the available bandwidth.

4.2 Experiments with Only Congestion Losses

In this section, we present the results from the simulations, where the losses are only due to congestion. The TCP-DCR protocol was designed with the goal of providing robustness to wireless channel errors, with minimal modifications to the core TCP behavior. Hence, in the absence of channel errors, we would like the TCP-DCR protocol to behave similar to the TCP-SACK protocol. In this section, we evaluate this issue at three different levels:

1. Flow level—throughput (relative fairness) when TCP-SACK and TCP-DCR flows compete with each other, time taken to relieve and reclaim bandwidth for sudden changes in available bandwidth and interaction with Web-like transfers.
2. Protocol level—Packet Delivery time and RTT estimation for individual flows.
3. The network level—average queue lengths and drop rates at the bottleneck link.

The topology used for these experiments is as shown in Fig. 9. The links between the sources and the router are high-capacity wired links with bandwidth 100 Mbps, delay 5 milliseconds and buffersize equal to the delay-bandwidth-product. The link between the router and the base station is the wired bottleneck link of capacity 10 Mbps and delay 5 milliseconds. The links between the base station and the receivers are wireless with capacity 1 Mbps, delay 20 milliseconds, and queue-length of 50 packets. Congestion level on the bottleneck link is modified by varying the buffersize on the link between the router and the base station. The receiver advertised window is set such that in the absence of congestion at the bottleneck link, the per-flow throughput does not exceed the wireless link capacity to ensure that the congestion happens only on the link between the router

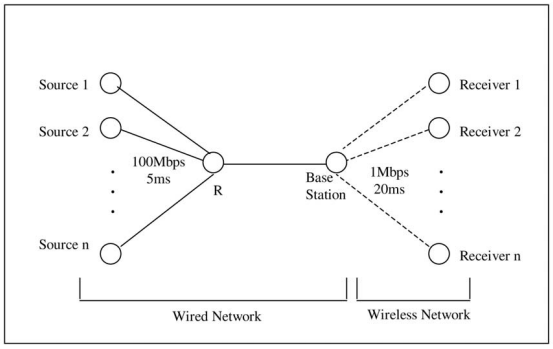

Fig. 9. Network topology for experiments with congestion losses.

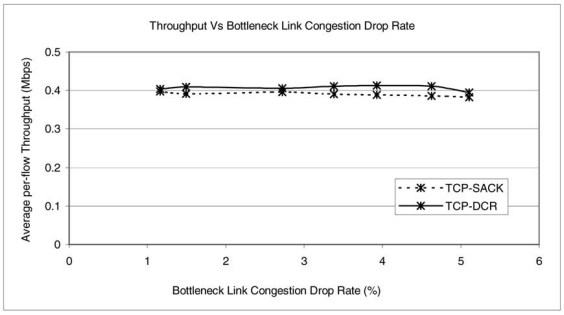

Fig. 10. Throughput versus congestion loss rate.

and the base station. Each source performs a single bulk data transfer to the corresponding receiver with a packet size of 1,000 bytes. The duration for the ftp transfer for most experiments in this paper is set to 1,100 seconds, but for the experiments inspecting the behavior at the flow level and the queue level, the transfer duration is smaller—200 seconds—due to the large amount of data being collected. The total number of flows in the network is 24 (unless otherwise mentioned).

4.2.1 Performance Comparison at Different Congestion Loss Rates

In this experiment, we evaluate the interaction between 12 TCP-DCR and 12 TCP-SACK flows. Fig. 10 shows the average throughput of the TCP-DCR flows in comparison with the average throughput of the TCP-SACK flows.

As can be seen from the graph, the TCP-DCR flows share the bottleneck link with the TCP-SACK flows in a relatively fair manner. For long-term flows, delaying the congestion response by one RTT does not make TCP-DCR more aggressive compared to the TCP-SACK flows. TCP-DCR is observed to respond to congestion faster than some of the other proposed protocols [16], [18] 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 [17]. Since TCP-DCR responds to congestion faster than these earlier protocols, we expect TCP-DCR will be safe even in dynamic network conditions.

4.2.2 Performance Comparison for Sudden Changes in Available Bandwidth

In this experiment, we evaluate the performance of TCP-DCR in comparison with TCP-SACK for sudden changes in

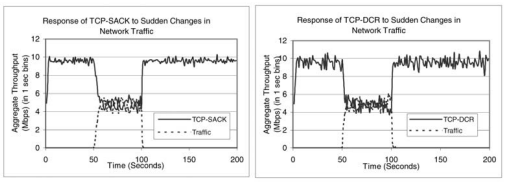

Fig. 11. Throughput versus time for sudden changes in traffic.

the available bottleneck bandwidth. The network consists of 24 flows. Half the flows do long-term ftp transfer starting at time 0 seconds using the protocol being evaluated. The other half of the flows carry shorter ftp transfer (referred henceforth as traffic) using TCP-SACK starting at 50 seconds and lasting for 50 seconds. Thus, 50 seconds after the long-term flows are started, the available network bandwidth goes down by 50 percent. At 100 seconds, the traffic stops, and the available bandwidth, doubles back to the original level. The average link droprate over the period of the simulation is about 2 percent. Fig. 11 shows the aggregate throughput of the long-term flows and the traffic (computed with 1 second bins) against time. From the figure, it is clear that the response of TCP-DCR to sudden fluctuations in traffic is similar to that 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 percent of the link capacity, thus allowing the new flows to achieve 45 percent of the link capacity. The time to reach (55 percent, 45 percent) allocation for TCP-SACK was 5.89 seconds and for TCP-DCR, it was 3.80 seconds. This shows that TCP-DCR is not worse than TCP-SACK in responding to sudden increases in traffic load.

4.2.3 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 eight 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 pseudorandom 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 percent. Fig. 12 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 simulation period is 4.76 Mbps, and that for the Web traffic is 4.84 Mbps. The aggregate throughput of TCP-DCR flows is 4.73 Mbps and that for the Web traffic is 4.82 Mbps. This indicates that the interaction of the TCP-DCR flows with short-term Web traffic is similar to that of TCP-SACK.

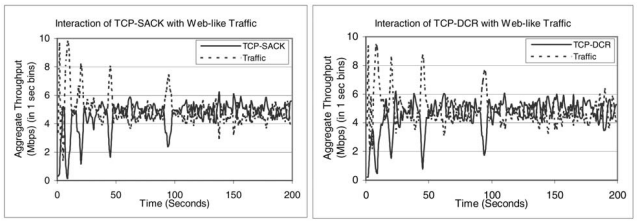

Fig. 12. Interaction with Web-like traffic.

4.2.4 Packet Delivery Time

In this section and the next, we take a look at some of the protocol-level dynamics. Since the TCP-DCR protocol delays the triggering of the congestion recovery algorithms by one RTT, it is possible that the packet delivery time during congestion is increased by upto one RTT. When there is no congestion in the network, the packet delivery time is unaffected. In this section, we present the results of simulations verifying the packet delivery time for the TCP-DCR flows in comparison to the TCP-SACK flows. Three separate simulations are considered—in the first, all 24 flows are TCP-SACK, in the second all 24 flows are TCP-DCR, and, in the third, half the flows (i.e., 12 flows) are TCP-SACK and the other half are TCP-DCR. This allows us to compare the packet delivery time for TCP-DCR with that of TCP-SACK, and also examine the effect of TCP-DCR flows on the packet delivery time of TCP-SACK flows when the workflows consists of a mix of the two flavors. The average congestion droprate at the bottleneck link is maintained at about 3.3 percent by using a buffersize of 70 packets at the bottleneck link. Fig. 13 shows the plot of packet delivery times for a randomly chosen TCP-DCR/TCP-SACK flow against the packet sequence number.

The plots show that the packet delivery times are scattered in two regions. The dense population of points around 60-100 milliseconds 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 the first simulation where all the flows are TCP-DCR, the average packet delivery time for packets of the sample flow recovered via retransmission is 398 milliseconds. In the second simulation where all the flows are TCP-SACK flows, it is 302ms. In the third simulation where 50 percent flows are TCP-DCR and the other 50 percent are TCP-SACK, the average packet delivery time for retransmitted packets of the sample TCP-DCR flow is 356 milliseconds and for

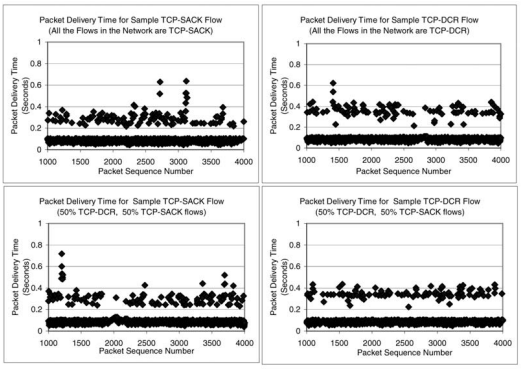

Fig. 13. Packet delivery time.

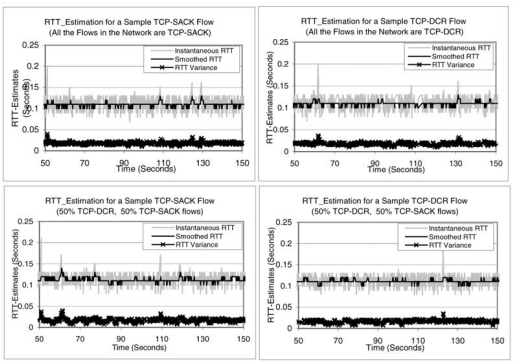

Fig. 14. RTT estimation.

TCP-SACK, it is 296 milliseconds. We notice from these observations that the recovery time for a retransmitted packet in case of the sample TCP-DCR flow is about one RTT more than that of the sample TCP-SACK flow. Also, we notice that when the workload consists of a mix of TCP-DCR and TCP-SACK flows, the time to recover a packet through retransmissions for TCP-SACK is not affected, compared to the simulation with all TCP-SACK flows.

4.2.5 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. 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. 14 shows the plot of instantaneous rtt, smoothed rtt, and rtt variance for a randomly chosen TCP-DCR/TCP-SACK flow against the packet sequence number. The results agree with the discussion presented here.

4.2.6 Effect on Network Queue Lengths

In this section, we evaluate the effect of TCP-DCR flows on the bottleneck link queue length. The network topology is similar to that in the above section. The average bottleneck link drop rate is about 3.3 - 3.4 percent. Fig. 15 shows the plot of the instantaneous and the average queue length at the bottleneck link.

With 24 flows in the network, the droptail queue at the bottleneck link is almost full all the time irrespective of whether the flows are TCP-DCR or TCP-SACK. Thus, it is hard to evaluate the impact of TCP-DCR on the queue lengths. The average queue length varies slightly (51 packets when all flows are TCP-DCR, 50 packets when all the flows are TCP-SACK, and 52 packets for the mixed workload), but the difference is negligible.

To further investigate this matter, we replaced the queue management scheme at the bottleneck link router with RED. The *minthresh_* and *maxthresh_* parameters are set to 25 and 75 percent of the total buffersize. Fig. 16 shows the plot of the instantaneous and the average queue length at the bottleneck link.

It can be seen from this graph that the queue length does not change much. The average queue lengths are 36, 34, and 35 packets, when all flows are TCP-DCR, TCP-SACK, or a mixture of the two, respectively.

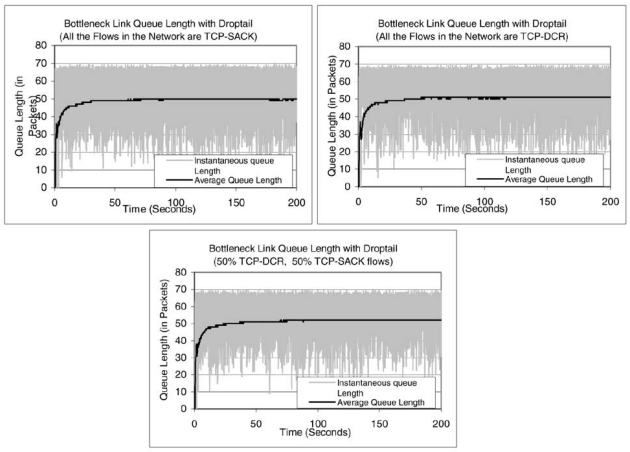

Fig. 15. Bottleneck link queue length with droptail queue management.

4.2.7 Effect on Bottleneck Link Congestion Loss Rate

One of the primary concerns when protocol characteristics are modified is the effect the modifications have on the network. TCP-DCR delays the response to loss notification. Hence, it is interesting to study how an increase in the offered load effects the congestion droprate on the bottleneck link. For this simulation, we keep all the other parameters constant and vary the number of flows in the network and study the congestion droprate at the bottleneck link. Note that the receiver window is adjusted such that the per-flow throughput is always less than the capacity of the wireless link and, hence, the congestion occurs only at the bottleneck link. The buffersize at the bottleneck link between the router and the base station is fixed at 50 packets to ensure that a wide range of congestion droprates may be observed, as the number of flows is varied. The simulations were conducted across the three traffic workloads considered in the earlier sections. The first graph in Fig. 17 shows the results. TCP-SACK (100, 0), TCP-DCR (0, 100) represent the average link droprate when all the flows in the network are TCP-SACK and TCP-DCR, respectively. TCP-SACK (50, 50) and TCP-DCR (50, 50) represent the average droprates observed by TCP-SACK flows and TCP-DCR flows, respectively, when the workload consists of a mix of both the flows. It can be seen from

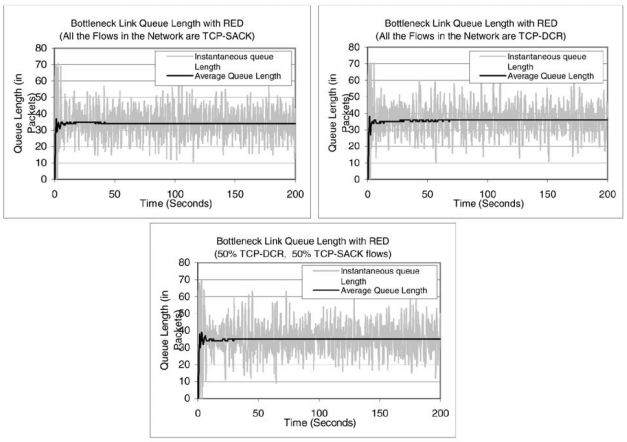

Fig. 16. Bottleneck link queue length with RED queue management.

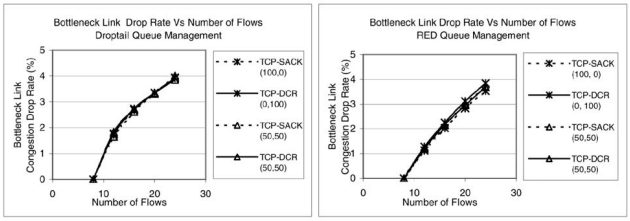

Fig. 17. Bottleneck link congestion loss rate versus number of flows.

the graph that the average congestion loss rate observed for TCP-DCR is similar to that of TCP-SACK.

Again, in the interest of being comprehensive, we repeated this experiment with RED queue management scheme at the bottleneck link. The second graph in Fig. 17 shows the results. In the previous section, we noticed that the average queue length is slightly different in the three cases. In an RED queue, the drop probability depends on the average queue length and, hence, the average drop probability varies slightly for the three cases, but the difference is fairly negligible.

4.3 Other Experiments

In this section, we present results for the simulations where the network has both wireless channel errors and congestion losses as well as results for comparison with TCP-Westwood. The topology for these networks is similar to that of experiments with congestion losses only.

4.3.1 Performance Comparison at Different Channel Error Rates and Congestion Losses

In this section, we present the results when the network has both channel errors and congestion. The network has 24 flows and the buffersize at the bottleneck link router is modified to obtain different levels of congestion. Half of the flows use TCP-SACK and the other half use TCP-DCR. Fig. 18 shows the results. In the graph, congestion loss rates of less than 1 percent are labeled as low error, in the range of 2.5-3.5 percent are labeled as moderate congestion and greater than 3.5 percent are labeled 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 Section 4.1.1 that the TCP-SACK flows cannot utilize the

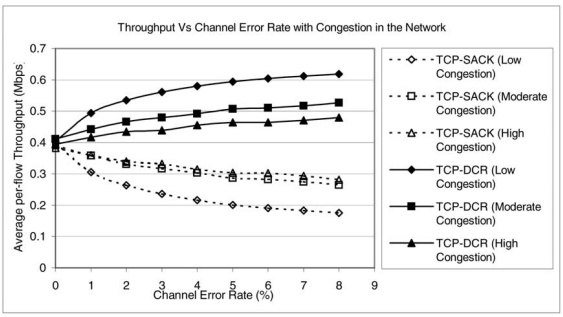

Fig. 18. Throughput versus channel error rate with congestion in the network.

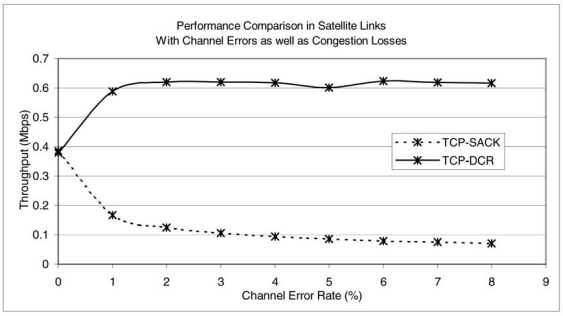

Fig. 19. Performance comparison over satellite links.

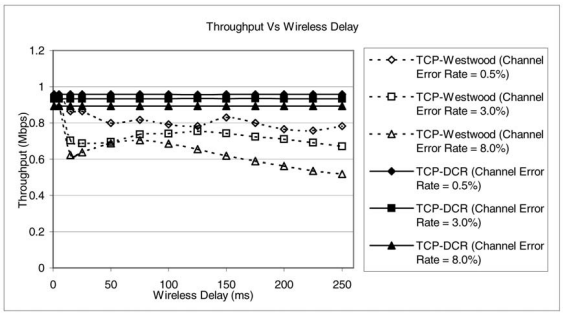

Fig. 20. Performance comparison of TCP-DCR versus TCP-Westwood.

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.

4.3.2 Performance Comparison on Satellite Links

Satellite links are characterized by very high wireless delays, with the one way delays being as large as 250 milliseconds [26]. With such high delays, when the congestion window is reduced unnecessarily in response to channel errors, it takes a long time to recover the window back to the optimal size. Thus, the performance of TCP-SACK degrades drastically in satellite networks as the channel error increases. In this section, we present the results of the simulations for performance comparison on satellite links. The network topology is similar to that above, except that the wireless link has a large one way delay of 250 milliseconds, making the end-to-end RTT 520ms. The average link droprate due to congestion is in the range of 0.1 - 0.4 percent. Fig. 19 shows the results, demonstrating the performance improvements with TCP-DCR.

4.3.3 Comparison with Other TCP Flavors

We have conducted extensive simulations to compare the performance of TCP-DCR with TCP-Reno and TCP-Westwood [15]. Our simulations show that the performance of TCP-DCR is much better than that of TCP-Reno in the presence of channel errors. 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. 20. The WestwoodNR agent was used in this simulation in the ns-2.26 version. The topology for this simulation is the same as explained in Section 4.1.2. 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 Conclusion and Future Work

In this paper, we have proposed Delayed Congestion Response to improve the performance of TCP over wireless networks that support link-level recovery mechanisms. The main advantage of the TCP-DCR protocol is the simplicity with which the scheme can be implemented. Since modifications need to be made only to the TCP at the sender, the deployment may be easier than other schemes that require modifications to network infrastructure, the receivers and the sender. The base station does not have to maintain any state other than that required for a rudimentary link-level retransmission scheme. We have implemented TCP-DCR on the Linux 2.4.x network stack and are currently evaluating it on a realistic testbed.

An interesting benefit of using TCP-DCR is that it provides inherent robustness against loss of degradation due to packet reordering in the network [23]. This has led us to further investigate the possibility of using TCP-DCR as a unified solution for recovering from different types of noncongestion events.

Acknowledgments

The authors would like to thank Dr. Sally Floyd for her invaluable comments and helpful suggestions. They would also like to thank Pradeep Kyasanur for his suggestions. This work is supported in part by US National Science Foundation (NSF) grants ANI-0196413, NSF ANI-9909229, Texas Higher Education Board, Texas Informatics and Telecom Infrastructure Task Force (TITF), and Intel Corp.

References

[1] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R.H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," *IEEE/ACM Trans. Networking*, 1997.

[2] A. Bakre and B.R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," *Proc. 15th. Int'l Conf. Distributed Computing Systems (ICDCS)*, May 1995.

[3] R. Yavatkar and N. Bhagawat, "Improving End-to-End Performance of TCP over Mobile Internetworks," *Proc. Workshop Mobile Computing Systems and Applications*, Dec. 1994.

[4] K. Brown and S. Singh, "M-TCP: TCP for Mobile Cellular Networks," *ACM Computer Comm. Rev.*, vol. 27, no. 5, 1997.

[5] K.-Y. Wang and S.K. Tripathi, "Mobile-End Transport Protocol: An Alternative to TCP/IP over Wireless links," *Proc. IEEE INFOCOM '98*, vol. 3, p. 1046, 1998.

[6] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz, "Improving TCP/IP Performance over Wireless Networks," *Proc. ACM MOBICOM*, Nov. 1995.

[7] H.M. Chaskar, T.V. Lakshman, and U. Madhow, "TCP over Wireless with Link Level Error Control: Analysis and Design Methodology," *IEEE Trans. Networking*, vol. 7, no. 5, Oct. 1999.

[8] D.A. Eckhardt and P. Steenkiste, "Improving Wireless LAN Performance via Adaptive Local Error Control," *Proc. IEEE Int'l Conf. Network Protocols (ICNP),* Oct. 1998.
[9] H. Balakrishnan and R.H. Katz, "Explicit Loss Notification and Wireless Web Performance," *Proc. IEEE GLOBECOM,* Nov. 1998.
[10] K. Ramakrishnan and S. Floyd, "A Proposal to Add Explicit Congestion Notification (ECN) to IP," RFC 2481, Jan. 1999.
[11] 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, Feb. 2002.
[12] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bhargavan, "WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks," *Proc. ACM MOBICOM,* Aug. 1999.
[13] N.H. Vaidya, M. Mehta, C. Perkins, and G. Montenegro, "Delayed Duplicate Acknowledgement: A TCP-Unaware Approach to Improve Performance of TCP over Wireless," *J. Wireless Comm. and Mobile Computing,* special issue on reliable transport protocols for mobile computing, Feb. 2002.
[14] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, Oct. 1996.
[15] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," *Proc. ACM MOBICOM,* 2001.
[16] D. Bansal and H. Balakrishnan, "TCP-Friendly Congestion Control for Real-Time Streaming Applications," *Proc. INFOCOM '00,* Apr. 2000, also available at "Binomial Congestion Control Algorithms," *Proc. IEEE INFOCOM,* 2001.
[17] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker, "Dynamic Behavior of Slowly Responsive Congestion Control Algorithms," *Proc. ACM SIGCOMM,* Sept. 2001.
[18] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," *Proc. ACM SIGCOMM,* 2000.
[19] M. Mathis, J. Semske, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithms," *ACM Computer Comm. Rev.,* vol. 27, no. 3, July 1997.
[20] T.V. Lakshman and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss," *IEEE/ACM Trans. Networking,* vol. 5, no. 3, June 1997.
[21] ns-2 Network Simulator, http://www.isi.edu/nsnam/, 2004.
[22] R. Han and D.G. Messerschmitt, "A Progressively Reliable Transport Protocol For Interactive Wireless Multimedia" *ACM/Springer-Verlag Multimedia Systems J.,* vol. 7, no. 2, Mar. 1999.
[23] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "RR-TCP: A Reordering-Robust TCP with DSACK," ICSI Technical Report TR-02-006, Berkeley, Calif., July 2002.
[24] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581, Apr. 1999.
[25] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit," RFC 3042, Proposed Standard, Jan. 2001.
[26] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP over Satellite Channels Using Standard Mechanisms," RFC 2488, Jan. 1999.
[27] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135, June 2001.

Sumitha Bhandarkar received the BE degree in electrical engineering from Sri Jayachamarajendra College of Engineering, Mysore University, India, in 1997 and the MS degree in computer engineering from the Texas A&M University, College Station, Texas, in 2001. She was awarded the National Merit Scholarship from the Government of India during 1991-1997 and she worked as a senior software engineer at Tata Unisys Ltd. (India) from 1997-1999. Currently, she is pursuing a PhD at the Texas A&M University. Her research interests are in the areas of high-performance networking, Internet congestion control, and network security. She is a student member of the ACM and the IEEE Computer Society.

Nauzad Erach Sadry received the BE degree in computer engineering from Bombay University, India, in September 1999. In Dec. 2002, he received the MS degree in computer science from Texas A&M University. He was awarded The Dorab Tata and J.N. Tata Merit Scholarships for Academic Excellence in 2000, India. Currently, he works as a software engineer at Z-Force Incorporated, Laguna Hills, California, where he is engaged in developing a high-performance file switch that aggregates multiple NAS devices. His research interests include wireless networking, storage area networks and distributed processing systems.

A.L. Narasimha Reddy (M'87-SM'97) received the BTech degree in electronics and electrical engineering from the Indian Institute of Technology, Kharagpur, India, in August 1985 and the MS and PhD degrees in computer engineering from the University of Illinois at Urbana-Champaign in May 1987 and August 1990, respectively. At the University of Illinois at Urbana-Champaign, he was supported by an IBM Fellowship. He is currently an associate professor in the Department of Electrical Engineering at Texas A&M University. He was a research staff member at IBM Almaden Research Center in San Jose from August 1990 to August 1995. Dr. Reddy's research interests are in network security, network QoS, multimedia, I/O systems, and computer architecture. Currently, he is leading projects on building scalable multimedia storage servers and partial-state-based network elements. While at IBM, he coarchitected and designed a topology-independent routing chip operating at 100 MB/sec, designed a hierarchical storage management system and participated in the design of video servers and disk arrays. Dr. Reddy is a member of ACM SIGARCH and is a senior member of IEEE Computer Society. He received a US National Science Foundation CAREER Award in 1996. He received an outstanding professor award at Texas A&M University during 1997-1998 and 2003-2004

Nitin H. Vaidya received the PhD degree from the University of Massachusetts at Amherst. He is presently an associate professor of electrical and computer engineering at the University of Illinois at Urbana-Champaign (UIUC). He has held visiting positions at Microsoft Research, Sun Microsystems, and the Indian Institute of Technology-Bombay. His current research is in the areas of wireless networking and mobile computing. His research has been funded by various agencies, including the US National Science Foundation, Defense Advanced Research Projects Agency, BBN Technologies, Microsoft Research, and Sun Microsystems. Dr. Vaidya is a recipient of a CAREER award from the US National Science Foundation. He has served on the program committees of several conferences and workshops, and served as program cochair for the 2003 ACM MobiCom. He has served as editor for several journals, and presently serves on the *IEEE Transactions Mobile Computing* editorial board, and as editor-in-chief of *ACM SIGMOBILE* periodical MC2R. He is a senior member of the IEEE Computer Society and a member of the ACM. For more information, please visit http://www.crhc.uiuc.edu/~nhv/.