Internet Engineering Task Force
INTERNET DRAFT
draft-ietf-tcpm-tcp-dcr-01.txt
Expires : February 2005

Sumitha Bhandarkar
A. L. Narasimha Reddy
Texas A&M University
August 2004

Improving the robustness of TCP to Non-Congestion Events.

Status of this Memo

Abstract:

This document proposes TCP-DCR, a simple modification to the TCP congestion control algorithm to make it more robust to non-congestion events. In the absence of explicit notification from the network, the TCP congestion control algorithm treats the receipt of three duplicate acknowledgements as an indication of congestion in the network. This is not always correct, notably so in wireless networks with channel errors or networks prone to excessive packet reordering, resulting in degraded performance. TCP-DCR aims to remedy this by delaying the congestion response of TCP for a short interval of time tau, thereby creating room to handle any non-congestion events that may have occurred. If at the end of the delay tau, the event is not handled, then it is treated as a congestion loss. The modifications themselves do not handle the non-congestion event, but rather rely on some underlying mechanism to do this. This document discusses the implications of delaying congestion response on the fairness, TCP-compatibility and network dynamics, and the benefits to be gained by applying the TCP-DCR modifications to TCP.

1. Introduction

In the absence of explicit notification from the network, the TCP sender treats the receipt of three duplicate acknowledgements (dupacks, for short) as an indication of congestion in the network. It responds by triggering the fast retransmit/fast recovery algorithm, where the packet perceived to be lost is retransmitted and the congestion window is reduced by half to relieve the congestion in the network. When the reason for the generation of dupacks is not congestion related, this reduction of the congestion window results in sub-optimal performance.

The two chief non-congestion events that might cause the generation of dupacks considered in this document are channel errors in wireless networks and excessive packet reordering. Several different solutions have been proposed in literature to improve the performance of TCP in the presence of channel errors [BB95,BPSK97,BS97,BSAK95,CLM99,MCGSW01,SVSB99,VMPM02,WT98,YB94] or packet reordering[BA02,ZKFP02]. This document proposes TCP-DCR which is a simple and unified solution to improve the robustness of TCP to any non-congestion event. Even though the discussion here is focussed on the two chief causes mentioned above, the solution is general enough to be extended to other non-congestion events resulting in the generation of dupacks.

Throughout the rest of this document, the term "TCP-DCR" is used to refer to the modifications that need to be made to TCP to make it robust to non-congestion events as well as to refer to the TCP flavor to which the modifications have been applied.

2. Problem Description

The strength of TCP lies in its ability to adjust its sending rate according to the perceived congestion in the network. In the absence of explicit notification of congestion from the network, the traditional TCP flavors use the loss of a packet as an indication of congestion. In order to help the sender identify a lost packet the receiver sends acknowledgements for every packet received in-order and duplicate acknowledgements (dupacks) for every packet received out-of-order. The acks were specified originally in order to clock out new packets. The use of three dupacks as an indication of congestion was added later. When the sender receives three consecutive dupacks, it concludes that the packet is lost due to congestion.

The TCP sender does not respond to the very first dupack, but waits for three dupacks to allow for a mildly reordered packet to reach the receiver, and possibly result in a cumulative acknowledgement.

draft-ietf-tcpm-tcp-dcr-01 August 2004

Limited Transmit, which is now Proposed Standard, allows the sender to send new packets in response to the first and second dupacks. The choice of waiting for three dupacks is purely heuristic. When the network is responsible for non-negligible amounts of non-congestion events, this trigger of three dupacks tends to be short and drastic. The persistent occurance of non-congestion events causes the TCP sender window to oscillate around a smaller value than what is actually allowed by the congestion in the network, resulting in degraded performance.

It is interesting at this point to review the prevalence of non-congestion events on the Internet. The two chief causes that are identified and targeted in this document are - wireless channel errors, and packet reordering within the network. While the existence of channel errors in the wireless networks is a well accepted fact, there is a general perception that packet reordering within the Internet is a rare phenomenon. Several recent measurement studies [BPS99,JIDKT03] though have shown results contrary to this popular sentiment. Even if we were to suppose that the amount of packet reordering in the current Internet is negligibly small, the need for almost in-order packet delivery places a severe constraint on the design of novel routing algorithms, network components and applications. For instance, high speed packet switches could cause resequencing of packets and there has been work proposed in the literature to ensure that packet ordering is maintained in such switches [KM02]. Other examples are multi-path routing, high-delay satellite links and some of the schemes proposed for differentiated services architecture. By making TCP more robust to non-congestion events, we aim to ease this restriction of always in-order delivery on the design of the future Internet components.

3. Design Guidelines

The proposal for TCP-DCR in this document is motivated by the following requirements -

* Improve the robustness of TCP to non-congestion events in general, rather than on a case-by-case basis.

* Maintain the end-to-end TCP semantics.

* Require a minimal amount of modification to the network infrastructure.

* The solution should lend itself to incremental deployment.

* After the modifications, the protocol should remain compatible with existing flavors of TCP.

4. Modifications to TCP

The TCP-DCR modifications involve simple changes regarding when the fast retransmit/recovery algorithms should be triggered. The current TCP flavors wait for three dupacks before responding as if a packet is lost due to congestion. This document extends the concept further by allowing the TCP-DCR sender to wait for an interval of tau after receiving the first dupack before responding to it as if it were a packet lost due to congestion. During the period tau, the TCP sender sends one new packet for every incoming dupack, if the congestion window allows it, similar to what is proposed by the Limited Transmit algorithm [ABF01]. The sender also continues to increase the congestion window during this period. However, since only one packet is allowed to be sent in response to each dupack, the number of packets on the link at any point remains the same as (or less than) the number of packets on the link when the first dupack was received.

The following figure illustrates the behavior of TCP in the presence of packet reordering, when the TCP-DCR modifications are applied.

```
                                           |<-------- tau -------->|
                                           Cong Response Delay Timer
                                      Limited Transmit/Additive Increase

          No Retransmission/Window Reduction ----+
                                                 |
                   Set Cong Response ------+     |   Cong Resp Delay
                       Delay Timer         |     |   Timer Cancelled
                                           |     |             |
           | <-- Round Trip Time --> |     v     v             |
                                                               |
           1  2  3  4  5  6          7     8  9  10  11        v
Sender  ---,--,--,--,--,--,----------,-----,--,--,--,----------,-------
            \  *  \  \  \  \        / \   / \/ \/ \/ \        /
             \    *\  \  \  \      /     /  /  /  /          /
              \     \* \  \  \    /        /                /
               \     \  *  \  \  /     /  /  /  /          /
                \     \  \ *\  \/     /  /  /  /          /
                 \     \  \  \*/\    /  /  /  /          /
                  \     \  \  \  *  /  /  /  /          /
                   \     \  \/ \  \/* /  /  /          /
                    \     \ /\  \ /\ / */  /          /
                     \     \  \  \  \  /  *          /
                      \   / \  \/ \/ \/  /   *      /
                       \ /   \ /\ /\ /\ /       *  /
Rcvr    ----------------`-----`--`--`--`----------*--------------------
                        2     2  2  2  2           8
```

Figure 1: Behavior of TCP-DCR in the presence of packet reordering.

As it can be seen from the figure, when the first dupack is received, the congestion response delay timer is set. When three dupacks are received, if the congestion response delay timer has not expired, the fast retransmit/recovery algorithm is not triggered. If the acknowledgement for the reordered packet reaches the sender before the delay timer expires, then the timer is cancelled and the sender does not suffer unnecessary reduction in the sending rate.

The following figure illustrates the behavior of TCP in the presence of packet loss due to congestion, when the TCP-DCR modifications are applied.

```
                                   | <-------- tau ---------> |
                                    Cong Response Delay Timer
                                         Limited Transmit
                                     Additive Window Increase

            No Retransmission ------------+
            No Window Reduction           |
                                          |
            Set Cong Response ------+     |
               Delay Timer          |     |   Retransmission -+
                                    |     | Window Reduction  |
                                    |     |                   |
    | <-- Round Trip Time --> |     v     v                   v

    1  2  3  4  5  6          7     8  9  10  11        12    2
 ---,--,--,--,--,--,----------,-----,--,--,--,----------,-----,--,--,--
     \  \  \  \  \  \        / \   / \/ \/ \/ \        / \   / \/  /
      \  \  \  \  \  \      /   \ /  /\ /\ /\  \      /     /  /  /
       \  \  \  \  \  \    /     \  /  \  \  \  \    /     /  /  /  /
        \  \  \  \  \  \  /     / \/  / \/ \  \  \  /     /  /  /  /
         \  \  \  \  \  \/     /  /\ /  /\  \  \  \/     /  /  /  /
          \  \  \  \  \ /\    /  /  /  /  \  \  \ /\    /  /  /  /
Cong Drop --> X  \  \  \  \  /  /  /  \/    \  \  \  \  /  /  /  /
            \     \  \/ \  \/  /  /  /\     \  \/ \  \/  /  /  /
             \     \ /\  \ /\ /  /  /  \     \ /\  \ /\ /  /  /
              \     \  \  \  \  /  /    \     \  \  \  \  /  /
               \   / \  \/ \/ \/  /      \   / \  \/ \/ \/  /
                \ /   \ /\ /\ /\ /        \ /   \ /\ /\ /\ /
 ----------------`-----`--`--`--`----------`-----`--`--`--`-----------
                 2     2  2  2  2          2     2  2  2  2
```

Figure 2: Behavior of TCP-DCR in presence of packet loss due to congestion.

The figure above shows the behavior of a TCP flow with the TCP-DCR modifications when a packet has been dropped due to congestion in the

network. In this case a cumulative acknowledgement is not received before the congestion delay timer expires. As a result, as soon as the congestion delay timer expires, the fast retransmit/recovery algorithm is triggered. The next section discusses the upper threshold on the delay tau so that this delay in congestion response does not adversely affect the throughput obtained by the flow using TCP-DCR modifications or the non TCP-DCR flows competing with it.

4.1. Choice of the delay duration (tau)

The current implementations of TCP wait for three dupacks before treating them as an indication of packet loss due to congestion. The choice of waiting for three dupacks is heuristic. This document proposes that the delay before responding to congestion should be longer, so that underlying schemes have time to recover from non-congestion events. There is no optimal value for this delay such that all possible non-congestion events can be recovered. It is essentially a tradeoff between unnecessarily inferring congestion, and unnecessarily waiting for a long time before retransmitting a lost packet. Therefore, the choice of the delay is really choosing a place on the spectrum for the tradeoffs between these two concerns. This document aims to provide guidelines for reasonable bounds on the delay to make it useful, without adversely modifying the TCP behavior.

Consider the case of wireless channel errors. The figure below shows a general scenario where the TCP sender is connected to the base station by a wired link and the TCP receiver is connected to a base station over a wireless link. The wired path between the base station and the sender TCP 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'.

```
                                   +----------------+
                                   |       rtt      |
                                   |                |
                       wired       |     wireless   |
            TCP        link        V       link     |     TCP
           Sender  0---------------0----------------0  Receiver
                   ^             Base               |
                   |            Station             |
                   |                                |
                   |              RTT               |
                   +--------------------------------+
```

Figure 3: General scenario for a wireless network.

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 an 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, to allow the link layer to recover the packet. In practice, the inter-packet delays are non-zero and the TCP sender does 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 upper bound on the delay is imposed by the retransmission timer of TCP. The delay should be chosen such that the RTO timeout is avoided, because a timeout would be detrimental to the performance of protocol. The RTO is usually set to (RTT + 4 * RTTVAR). The standard recommends a minimum of 1 second, but many TCP implementations have a much smaller minimum, e.g., 100 ms. This forms the upper bound on the value for the congestion response delay tau.

Based on the above discussion, this document recommends the value of tau to be set as 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 tau, that will facilitate the recovery of a packet reordered by any amount. However, the upper bound is still decided by the discussion above. So, a value of one RTT for tau is still a reasonable choice. We conducted the analysis of the steady state bandwidth realized by TCP-DCR [BR03]. The results of the analysis show that the TCP-DCR modifications do not affect the steady state bandwidth.

TCP-DCR does not increase the per-packet delivery time when there is no congestion in the network. However, when a packet is dropped, the choice of tau = one RTT may add upto one additional RTT of delay in recovering the lost packet. An important fact to remember here is that, the choice of tau does not cause the TCP-DCR sender to dramatically over-send packets because the protocol is still ACK-clocked. That is, a new packet is sent only upon the receipt of a dupack. If there is suddenly very high congestion in the network resulting in the drop of several packets, the TCP sender will have reduced its sending rate simply because not many dupacks are coming back.

4.2. Implementation Details

The TCP-DCR modifications need to be applied only to the sender and the receiver remains unmodified. The sender can implement the delay in congestion response (tau) by using either a timer or by modifying the threshold on the number of duplicate acknowledgements to be received before triggering fast retransmit/recovery. The timer-based implementation is quite straight forward, but is influenced by the coarseness in the clock granularity. In the ack-based delay implementation, the sender could delay responding to congestion for the number of duplicate acknowledgements corresponding to the delay required. Thus, if 'tau' is chosen to be one RTT, the sender would wait for the receipt of 'W' duplicate acknowledgements before responding to congestion, where 'W' is the size of the congestion window when the packet loss is detected.

The TCP-DCR modifications work with most flavors of the TCP protocol. However, this document advocates 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 congestion related 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 NewReno.

4.3. 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 'tau', 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, 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 'tau' 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 'tau' allowing the local recovery mechanism to recover from non-congestion event.

4.4. Underlying mechanisms for recovering from non-congestion events

The performance benefits to be gained from using the TCP-DCR modifications depends heavily on the existence of an underlying scheme for recovering from the non-congestion events. In the case of packet reordering, no explicit scheme is required to recover the reordered packet; the reordered packet reaches the receiver after the delay that caused it to appear out-of-order. In the case of wireless networks, a packet corrupted due to channel errors might be recovered through link-level mechanisms such as link-level retransmissions or FEC (Forward Error Correction). If the corrupted packet is not recovered through link-level mechanisms, it will be interpreted by TCP as a packet lost due to congestion, and retransmitted by TCP.

5. Performance Evaluation

This section of the document provides a glimpse of the performance improvements to be gained by the use of TCP-DCR modifications. The results presented here are only a small subset of the results presented in [BR03]. The results are based on simulations on the ns-2 simulator [NS-2].

5.1. Network with packet reordering

The table below shows the effect of delayed packets on the performance of TCP-SACK and the corresponding improvement in the performance in case of TCP-DCR. The experiment is conducted with a dumbell topology with the bottleneck link bandwidth set to 8Mbps. The end-to-end RTT is set to 104ms. 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 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 throughput of TCP-SACK without the TCP-DCR modifications degrades drastically. However, when the TCP-DCR modifications are applied the performance is very good even when a large percentage of the packets are delayed.

Percentage of Packets Delayed (%)	Throughput of TCP-SACK without TCP-DCR modifications (Mbps)	Throughput of TCP-SACK with TCP-DCR modifications (Mbps)
0.0	7.325	7.352
1.0	1.043	7.339
2.0	0.795	7.309
5.0	0.571	7.185
8.0	0.498	7.095
10.0	0.476	7.061
15.0	0.440	7.000
20.0	0.410	7.008
25.0	0.409	7.014
30.0	0.404	7.006

5.2. Wireless Networks with Channel Errors

The table below shows the effect of channel errors on the performance of TCP-SACK with and without the TCP-DCR modifications. The topology for the experiment consists of a sender connected via a wired link to a router which in turn is connected to the base station by a wired link. The bandwidth of the wired links is 100Mbps and the delay is 5ms. The receiver is connected to the base station by a link simulating a satellite connection with a lower bandwidth and a larger delay. The bandwidth of this link is 1Mbps and the delay is 250ms. Packets are randomly chosen to be corrupted by channel errors. Link level retransmission is simulated by retransmitting the corrupted packet after a delay corresponding to the round trip time of the wireless link.

Channel Error Rate (%)	Throughput of TCP-SACK without TCP-DCR modifications (Mbps)	Throughput of TCP-SACK with TCP-DCR modifications (Mbps)
0.0	0.962	0.962
0.5	0.261	0.957
1.0	0.186	0.952
2.0	0.131	0.943
3.0	0.107	0.934
4.0	0.094	0.925
5.0	0.086	0.917
6.0	0.081	0.908
7.0	0.078	0.900
8.0	0.073	0.892

5.3. Fairness Implications

This section of the document addresses the fairness issues raised by delaying congestion response. The steady state analysis of TCP-DCR [BR03] shows that the throughput of the TCP-DCR protocol is similar to that of TCP [PFTK98]. Thus, the congestion control dynamics of

TCP-DCR are TCP-friendly. Essentially, TCP-DCR can be seen as a slowly-responsive TCP-friendly flow as explained in [BBFS01]. It has been shown in that paper that such flows are TCP-compatible.

Simulation results agree with the discussion above. The following table shows the average throughput achieved by flows using TCP-SACK without the TCP-DCR modifications compared to flows using TCP-SACK with the TCP-DCR modifications in a congested network. The dumbell topology is used for this experiment with the bottleneck link capacity of 10Mbps being shared by 12 flows, half of which are TCP-SACK without TCP-DCR modifications and the other half are TCP-SACK with the TCP-DCR modifications. There are no non-congestion losses in the network and congestion is induced by modifying the buffers available at the bottleneck router. The throughput of each individual flow varies only slightly from the average throughput.

Congestion Droprate (%)	Avg. Throughput of TCP-SACK without TCP-DCR Modifications (Mbps)	Avg. Throughput of TCP-SACK with TCP-DCR Modifications (Mbps)
0.06	0.808	0.795
0.36	0.820	0.782

1.51	0.837	0.765
1.86	0.828	0.774
2.44	0.836	0.767
3.43	0.767	0.835
4.57	0.724	0.874
5.76	0.719	0.788

5.4. Effect on Network Dynamics

When the loss of a packet is indeed due to congestion, delaying the congestion response could make the protocol sluggish at relieving congestion in the network. However, when the delay is bounded by one RTT, the behavior of TCP-DCR is not significantly different from a TCP flow with high variance in RTT measurements. During the congestion response delay, the TCP-DCR flow appears like a flow whose RTT is twice the value when there is no congestion in the network. Performance evaluation through simulations has validated this view [BR03].

6. Implementation Issues

The TCP-DCR modifications presented by this document are quite simple and do not require complicated changes. When the delay "tau" is implemented based on a timer, the timer value can be set to the

smoothed value of RTT (SRTT). However, when the delay "tau" is implemented by modifying the threshold on the number of dupacks to be received before responding, the RTT value being used is essentially the instantaneous value. The upper bound on the congestion response delay is established by the RTO estimate which is computed based on the smoothed RTT. This could potentially lead to a situation where the value of the congestion response delay is larger than the value of the RTO. Though such a situation could be fairly rare, even few unnecessary timeouts can degrade the performance drastically. So, this document recommends that the new threshold on the number of dupacks to wait before responding be scaled by the factor (SRTT)/(Current RTT Estimate).

We have implemented the TCP-DCR modifications in the Linux 2.4.20 kernel. The modifications require changes of only a few lines of code. Currently, we are in the process of evaluating the reordering robustness provided by native Linux implementations against that of TCP-DCR.

7. Incremental Deployment

The TCP-DCR modifications proposed in this document lend themselves to incremental deployment. Only the TCP protocol on the sender side needs to be modified. The modifications themselves are minor and can be distributed easily as kernel patches. The use of TCP-DCR does not require the sender and receiver to negotiate any conditions during connection setup. Neither the receivers nor the routers need to be aware that the sender has been enhanced with the TCP-DCR modifications. Availability of additional buffers at the receiver will help maximize the benefits of using TCP-DCR but are not necessary.

8. Relationship to other 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 [BB95,BS97,WT98,YB94], TCP-aware link layer protocols [BSAK95,CLM99], explicit loss notification approaches [BK98,KAPS02,RF99] and receiver-based approaches [SVSB99,VMPM02]. All the above mentioned schemes are proposed explicitly for improving the performance of TCP in wireless networks. While some of them could possibly be used in situations with other types of non-congestion events, the simplicity of TCP-DCR in our opinion, makes it a far more compelling solution for the problem.

It has been shown that the performance of TCP over wireless networks can be improved by using other flavors of TCP. For example, by using

TCP-SACK [MMFR96] or TCP-westwood [MCGSW01] instead of standard implementations of TCP Reno, performance can be improved. The performance improvement by using TCP-SACK protocol however, is due to its ability to recover from multiple losses in one RTT and does not necessarily indicate robustness to non-congestion events. This document advocates the use of TCP-DCR modifications with the TCP-SACK flavor.

Different solutions have been proposed in the literature to improve the performance of TCP when the network reorders packets persistently. In [BA02] the authors present several schemes which use DSACKs [FMMP00] (or could alternatively use timestamps [LM03] or other methods) to identify a false fast retransmit. In response, 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 threshold on the number of dupacks to be received before responding (dupthresh) is increased to avoid future 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 [ZKFP02] 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 solutions 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 Linux TCP provides an option of using either of these additional schemes or just the information from SACK to estimate the reordering length. The intent is to estimate the optimal amount of time to delay the triggering of fast retransmit/recovery algorithms 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. If the actual reordering within the network is less than one RTT, then no harm is done since no action is necessary when the packet is recovered. When the packet is reordered by more than one RTT, TCP-DCR does not wait for it it to be recovered, but in doing so avoids costly retransmission timeouts.

9. Security Considerations

This proposal makes no changes to the underlying security of TCP.

10. Conclusions

This document has proposed TCP-DCR modifications to TCP's congestion control mechanism to make it more robust to non-congestion events. We have explored this proposal though analysis and simulations, and are currently in the process of evaluating it through experiments on the Linux platform. We believe that TCP-DCR provides a simple, unified solution to improve the the robustness of TCP to non-congestion events, and that the solution is safe to deploy on the Internet. We would welcome additional analysis, simulations, and experimentation.

We are bringing this proposal to the IETF to be considered as an Experimental RFC.

11. Acknowledgements

We would like to thank Dr. Nitin Vaidya and Nauzad Sadry for their invaluable help with the wireless simulations. Comments from Sally Floyd have helped immensely in improving the quality of this document.

12. References

[ABF01] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit," RFC 3042, Proposed Standard, January 2001.

[BA02] E. Blanton and M. Allman, "On Making TCP More Robust to Packet Reordering," ACM Computer Communication Review, January 2002.

[BB95] 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.

[BBFS01] D. Bansal, H. Balakrishnan, S. Floyd and Scott Shenker, "Dynamic Behavior of Slowly Responsive Congestion Control Algorithms," Proceedings of ACM SIGCOMM, Sep. 2001.

[BK98] H. Balakrishnan and R. H. Katz, "Explicit Loss Notification and Wireless Web Performance," Proc. of IEEE GLOBECOM, Nov. 1998.

[BPS99] J. Bennett, C. Partridge, and N. Shectman, "Packet reordering is not pat hological network behavior," IEEE/ACM Transactions on Networking, December 1999.

[BPSK97] 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.

[BR03] Sumitha Bhandarkar, and A. L. N. Reddy, "TCP-DCR: Making TCP Robust to Non-Congestion Losses," Technical Report TAMU-ECE-2003-04, July 2003.

[BS97] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," ACM Computer Communications Review, vol. 27, no. 5, 1997.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir and R. Katz, "Improving TCP/IP performance over wireless networks," Proc. of ACM MOBICOM, Nov. 1995.

[CLM99] 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.

[FMMP00] Sally Floyd, Jamshid Mahdavi, Matt Mathis and Matt Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP," RFC 2883, July 2000.

[JIDKT03] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Measur ement and Classification of Out-of-Sequence Packets in a Tier-1 IP Backbone," Pr oceedings of IEEE INFOCOM, 2003.

[KAPS02] 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

[KM02] I. Keslassy and N. McKeown, "Maintaining packet order in twostage switche s," Proceedings of the IEEE Infocom, June 2002

[LM03] R. Ludwig and M. Meyer, "The Eifel Detection Algorithm for TCP," RFC 3522, April 2003.

[MCGSW01] 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.

[MMFR] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, "TCP selective acknowledgment options," Internet RFC 2018.

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

[RF99] K. Ramakrishnan and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP," RFC 2481, January 1999.

[SVSB99] 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.

[VMPM02] 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.

[WT98] 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.

[YB94] R. Yavatkar and N. Bhagawat, "Improving End-to-End Performance of TCP over Mobile Internetworks," Workshop on Mobile Computing Systems and Applications, December 1994.

[ZKFP02] 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.

13. Author's Addresses

Sumitha Bhandarkar
Dept. of Elec. Engg.
214 ZACH
College Station, TX 77843-3128
Phone: (512) 468-8078
Email: sumitha@tamu.edu
URL : http://students.cs.tamu.edu/sumitha/

A. L. Narasimha Reddy
Associate Professor
Dept. of Elec. Engg.
315C WERC
College Station, TX 77843-3128
Phone : (979) 845-7598
Email : reddy@ee.tamu.edu

URL : http://ee.tamu.edu/~reddy/