TCP-Peach+: Enhancement of TCP-Peach for Satellite IP Networks

Ian F. Akyildiz, *Fellow, IEEE*, Xin Zhang, and Jian Fang

***Abstract*—In this letter an improvement to TCP-Peach congestion control scheme, called TCP-Peach+, is proposed to further improve the goodput performance for satellite IP networks. In TCP-Peach+, two new algorithms, Jump Start and Quick Recovery, are proposed for congestion control in satellite networks. These algorithms are based on low priority segments, called NIL segments, which are used to probe the availability of network resources as well as error recovery. Simulation experiments show that TCP-Peach+ outperforms TCP-Peach in terms of goodput and achieves fair share of network resources as well.**

***Index Terms*—Congestion control, high bit error rates, long propagation delays, satellite IP networks, TCP-Peach+.**

I. Congestion Control in TCP-Peach+

TRADITIONAL congestion control schemes perform poorly in satellite networks due to high *bit error rates* (BER) and long propagation delays [2]. These problems were addressed in TCP-Peach [1]. In this letter, we present an enhancement for TCP-Peach, called TCP-Peach+, for satellite IP networks. TCP-Peach+ includes the following new schemes: *Jump Start* and *Quick Recovery* and two classical algorithms, *Congestion Avoidance* and *Fast Retransmit*. The TCP SACK option [6] is also adopted in TCP-Peach+ to improve the performance when multiple segments are lost from one window of data.

A. NIL Segments

In TCP-Peach [1], the *dummy segments*, are used to probe the availability of network resources to improve the performance when segment losses are due to link error instead of congestion. In TCP-Peach+, we introduce low priority segments, called *NIL segments* to probe the availability of network resources as well as error recovery. The generation algorithm for *NIL segments* is shown in Fig. 1. Low-priority segments are also used in [7] to improve TCP *Slow Start* Algorithm. However, they are different in generation, functionality and retransmission policy.

Because *NIL segments* carry unacknowledged information, they can be used by the receivers to recover lost segments. In satellite networks with high bit error rates, this solution is more efficient than dummy segments which carry duplicate information.

Manuscript received February 22, 2002. The associate editor coordinating the review of this letter and approving it for publication was Prof. J. Choe.This work was supported by NASA-Glenn Research Center under the Grant NAG3-2585.

The authors are with the Broadband and Wireless Networking Laboratory, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: ian@ece.gatech.edu; xinzhang@ece.gatech.edu; jfang@ece.gatech.edu).

Publisher Item Identifier 10.1109/LCOMM.2002.801317.

```
Generate_NIL_Segment()
    i=0;
    if a NIL segment is needed
        q=i mod Q; i=i+1 ;
        select the qth packet in the queue as the NIL segment;
    end;
    if the jth element in the queue is ACKed
        remove that packet from queue Q ;
        if (j<Q and Q>0)
            Q=Q-1;
        end;
    end;
    if a new packet is added to the queue Q
        add the new packet at the tail of the queue;
        Q=Q+1;
    end;
end;
```

Fig. 1. NIL segment generating algorithm.

NIL segments are low priority segments. If a router on the connection path is congested, then the *NIL segments* are discarded first. Thus, the transmission of *NIL segments* does not affect data segments. For each *NIL segment*, a NIL ACK carried by a low priority IP packet is transmitted back to the sender. The arrival of NIL ACKs at the senders are indications that there are unused resources in the network. Hence, the senders increase their transmission rate accordingly to make full utilization of network resources.

B. Jump Start Algorithm

In satellite networks, long propagation delays cause TCP *Slow Start* to be inefficient since its $cwnd$ increases slowly. In TCP-Peach [1], a new scheme called TCP *Sudden Start* is proposed to improve TCP *Slow Start*. *Dummy segments* are used to probe the availability of network resources and increase $cwnd$ quickly. To further improve the *Slow Start* Algorithm, in TCP-Peach+, we propose *Jump Start* using NIL segments which is described in I-A to probe the availability of network resources and recover errors at the same time. The algorithm is shown in Fig. 2.

In *Jump Start*, the TCP sender sets the congestion window, $cwnd$, to 1. After sending the first data segment, it transmits $(rwnd - 1)$ *NIL segments* generated as in Fig. 1 every $\tau = RTT/rwnd$, where $rwnd$ is the receiver window size. As a result, after one *round trip time*, the congestion window size $cwnd$ increases very quickly as the ACKs for NIL segments arrive at the sender.

C. Quick Recovery Algorithm

In TCP-Peach, *dummy segments* are transmitted in *Rapid Recovery* to resume $cwnd$ rapidly from decrement due to segment loss caused by link error. Although it solves the problem of

```
Jump_Start()
    cwnd=1; $\tau$=RTT/rwnd;
    send(Data_Segment);
    for ( i=1 to rwnd-1 )
        wait($\tau$); send(NIL_Segment);
    end;
end;
```

Fig. 2. The Jump Start algorithm.

throughput degradation in satellite networks over Fast Recovery [5], when the link error is high and multiple segment losses occur within one window of data, the throughput degradation is still large. So we propose *Quick Recovery* to recover from high link errors.

As the TCP SACK proposed in [4], we adopt the SACK option field in TCP-Peach+ to avoid retransmitting out of order segments received at the destination. At the sender, a data structure called *scoreboard* is maintained to update information about cumulatively ACKed and SACKed segments. During *Quick Recovery*, a variable called $pipe$ that represents the estimated number of segments outstanding in the network is maintained. The variable $pipe$ is incremented by one when the sender either sends a new segment or retransmits an old segment. It is decreased by one when the sender receives an ACK that reports the new data has been received at the receiver and left the pipe. Whenever a SACK arrives, the *retransmit timer* is also reset. When the ACK for the segment arrives which is transmitted right before *Quick Recovery* is entered, which ACKs all data that is outstanding before *Quick Recovery*, the sender exits *Quick Recovery* and begins congestion avoidance normally.

The parameters used in Quick Recovery are as follows.

- $HighAck$ is the sequence number of the highest cumulative ACK received at a given point.
- $HighData$ is the highest sequence number transmitted just before Quick Recovery begins.
- a *Duplicate ACK* is an ACK whose cumulative ACK is equal to $HighAck$ and conveys new SACK information.
- a *Partial ACK* is an ACK that increases the $HighAck$ value, but does not ACK all of the data up to $HighData$.
- a *Recover ACK* is the ACK for all data up to $HighData$.
- $ndupacks$ is the number of duplicate packets which trigger the TCP to Fast Retransmit and Quick Recovery phases, which is chosen to be 3.
- $maxburst$ limits the number of packets that can be send in response to a single incoming ACK. It is chosen to be 4 as proposed in [4].
- $amountacked$ is the number of data segments that ACKed by the receiver when an ACK arrives.
- $adsn$ is the number of NIL segments that the TCP sender is allowed to inject into the network.
- $adps$ is the number of allowed data segments to be sent.
- nps is the number of data segments which have been really sent.

The details of the *Quick Recovery* algorithm is shown in Fig. 3. When the sender has received $ndupacks$ SACKs, which indicate segment loss, the *Quick Recovery* then behaves

```
Quick_Recovery()
  cwnd=cwnd/2;
  adsn=cwnd;
  adps=0;
  END=0;
  pipe=2*cwnd-ndupacks+1;
  while (END=0)
    if (ACK_ARRIVAL)
      if(Duplicate ACK)
        pipe=pipe-1;
        update scoreboard;
      end;
      else if (Partial ACK)
      pipe=pipe-amountacked;
        update HighAck;
        update scoreboard;
      end;
      if (NIL ACK)
        cwnd=cwnd+1;
        adsn=adsn-1;
        update scoreboard;
      end;
      if (Recover ACK)
        update HighAck;
        clear scoreboard;
        END=1;
      end;
      adps=cwnd-pipe;
        nps=min(maxburst, adps)
      if (nps>0)
      send nps missing packets and/or new packets;
        pipe=pipe+nps;
      end;
      else if (adsn>0)
        send a NIL packet;
        adsn=adsn-1;
      end;
    end;
  end;
end;
```

Fig. 3. The Quich Recovery algorithm.

conservatively after Fast Retransmit: the sender halves its congestion window $cwnd$. It means that approximately half of the original window size of data will be transmitted during Quick Recovery phase. The variable $pipe$ is initialized as $2^*cwnd - ndupacks + 1$ since the original congestion window is 2^*cwnd and the sender has received $ndupacks$ SACKs right before *Quick Recovery*, each of which reports a new data segment left the pipe. The sender also retransmits a lost segment using *Fast Retransmit* algorithm. The number of NIL segments sent during *Quick Recovery* $adsn$ is equal to $cwnd$. This is because *Quick Recovery* will last roughly one RTT time. The ACKs for NIL segments sent during *Quick Recovery* will be received in *Congestion Avoidance* phase. If the segment loss is due to link error, $cwnd$ will be increased by one for each NIL ACK arrival. Thus, the congestion window reaches its value before the data segment loss was detected rapidly.

During *Quick Recovery* phase, upon a duplicate ACK arrives, which indicates one segment left the network, $pipe$ is reduced by one. When a partial ACK arrives, $pipe$ is reduced by $amountacked$ and $HighAck$ is also updated. $Amountacked$ is the exact number of segments acknowledged by a partial ACK [3]. It is used to accurately estimate the number of segments ACKed either by data ACKs or NIL ACKs. The arrival of a NIL ACK indicates that there are still unused resources in the network. Therefore, $cwnd$ is increased by one and $adsn$ is reduced by one. Finally, if a cumulative ACK acknowledges all data up to and cover $HighData$, *Quick Recovery* phase is terminated.

When an ACK is received during *Quick Recovery*, the *scoreboard* is updated to keep the latest information about missing segments. The *scoreboard* is cleared when *Quick Recovery* is exited. The number of segments which can be sent to the network, $adps$, is $cwnd - pipe$. In order to avoid injecting bursty segments into the network, $maxburst$ is set. When data segments are allowed to be sent, the missing segments are always transmitted before new data segments. If no data segments are allowed to be sent, a NIL segment is transmitted as long as $adsn > 0$.

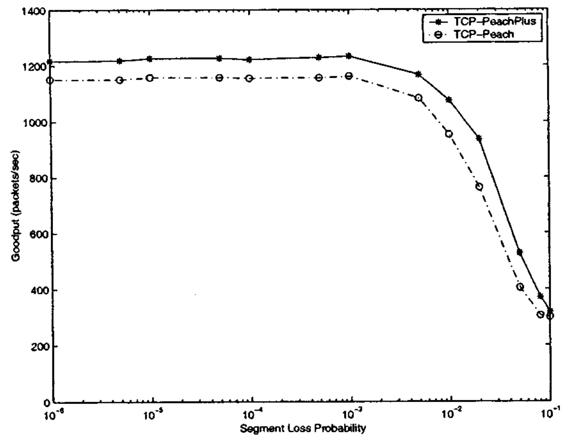

Fig. 4. Goodput performance comparison of TCP-Peach+ and TCP-Peach for different values of $\mathrm{P}_{\mathrm{Loss}}$.

II. Performance Evaluation

To evaluate the performance of TCP-Peach+, we simulate a GEO satellite system with N senders transmit data to N receivers. The N streams are multiplexed in the Earth station with buffer size of K. As in [1], we assume $N = 20$, $K = 50$ segments, $rwnd = 64$ segments, the link capacity $c = 1300$ segments/s which is approximately 10 Mb/s for TCP segments of 1000 bytes. The round-trip time $RTT = 550$ ms, the simulation time $t_{\mathrm{Simulated}} = 550$ s.

The goodput performance comparison of TCP-Peach+ with TCP-Peach for different values of segment loss probability P_{Loss} is shown in Fig. 4, where the goodput performance of TCP-Peach+ is better than TCP-Peach. This is due to *Jump Start* as well as *Quick Recovery*. When the loss probability is low, the goodput improvement is mostly because of *Jump Start* since the segment losses are mostly due to network congestion instead of link errors. In this case, *Nil segments* in TCP-Peach+ carry unacknowledged segments, which make the connections to recover from lost segments more rapidly than dummy segments in TCP-Peach. When the loss probability is high, the *Quick Recovery* contributes to the goodput improvement. It avoids halving the $cwnd$ multiple times due to multiple losses of segments within one window of data as in *Rapid Recovery*. In Fig. 4, when $P_{\mathrm{Loss}} = 5{*}10^{-2}$, the goodput improvement of TCP-Peach+ over TCP-Peach is 30.24%.

As dummy segments in TCP-Peach, NIL segments in TCP-Peach+ do not carry any new information. Thus, they introduce overhead in the network [1]. The overhead comparison of TCP-Peach+ versus TCP-Peach is depicted in Fig. 5. For most P_{Loss} values, the overhead ratio of TCP-Peach+ to TCP-Peach is less than one. This is because in *Quick Recovery* phase of TCP-Peach+, one NIL segment is sent when no data is allowed to be transmitted while in *Rapid Recovery* phase of TCP-Peach, two dummy segments are sent. For large P_{Loss} values which cause multiple segment losses within one window of data, $cwnd$ in *Quick Recovery* is reduced more slowly than in *Rapid Recovery*. Therefore, more NIL segments are allowed to be sent to probe network resources and recover errors.

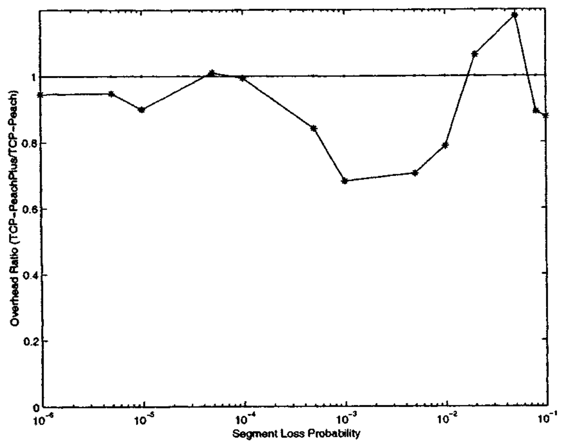

Fig. 5. Overhead comparison of TCP-Peach+ and TCP-Peach for different values of $\mathrm{P}_{\mathrm{Loss}}$.

When $P_{\mathrm{Loss}} = 5{*}10^{-2}$, the ratio is the maximum value, which indicates that the overhead of TCP-Peach+ is 18.03% higher than TCP-Peach. However, we already see that the goodput improvement in this case is 30.24%.

Besides goodput, we evaluate another important metric fairness of TCP-Peach+. We assume 10 connections $N = 10$ and $P_{\mathrm{Loss}} = 0$. The simulation results we obtained are consistent with the fairness evaluations in [1]. As a result, TCP-Peach+ provides a fair share of the network resources both in homogeneous and heterogeneous scenarios.

III. Conclusions

In this letter, we introduce a new congestion control protocol for satellite IP networks, TCP-Peach+, as an improvement over TCP-Peach [1]. A new type of low priority segment NIL segment is used to probe the availability of network resources as well as error recovery. Two new algorithms: *Jump Start* and *Quick Recovery* are adopted in TCP-Peach+ to recover rapidly from multiple segment losses within one window of data. The simulation results show that the performance of TCP-Peach+ is better than TCP-Peach and achieves fairness as well.

References

[1] I. F. Akyildiz, G. Morabito, and S. Palazzo, "TCP peach: A new congestion control scheme for satellite IP networks," *IEEE/ACM Trans. Networking*, pp. 307–321, June 2001.

[2] ——, "Research issues for transport protocols in satellite IP networks," *IEEE Pers. Commun.*, vol. 8, no. 3, pp. 44–48, June 2001.

[3] M. Allman and E. Blanton, "A conservative SACK-based loss recovery algorithm for TCP," *Internet Draft*, Jan. 2001.

[4] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," *ACM Comp. Commun. Rev.*, vol. 26, pp. 5–21, July 1996.

[5] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM'88*, Sept. 1988, pp. 314–319.

[6] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options,", RFC 2018, 1996.

[7] V. N. Padmanabhan and R. H. Katz, "TCP fast start: A technique for speeding up web transfers," in *Proc. IEEE Globecom'98 Internet Mini-Conf.*, Sydney, Australia, Nov. 1998.