A Reliable Multicast Transport Protocol for Satellite IP Networks

Jian Fang and Ian F.Ak yildiz
Broadband and Wireless Networking Laboratory
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30332.

Abstract—**In this paper, a multicast transport protocol, called TCP-Peachtree, is proposed for satellite IP networks. In addition to the acknowledgment implosion and scalability problems appeared in terrestrial wirelined networks, satellite multicasting has additional problems, i.e., low bandwidth feedback link, different multicast topology and congestion control problem. In TCP-Peachtree, the modified B+ tree hierarchical structure is used to form dynamic multicast groups. Local error recovery and ACK aggregations are performed within each subgroup and also via logical subgroups. Two new algorithms, Jump Start and Quick Recovery, which are based on a type of low priority segments, called NIL segments, are proposed for congestion control. NIL segments are used to probe the availability of network resources and also for error recovery. Moreover,an ACK filter is also introduced to aggregate ACKs. The simulation results show that the congestion control algorithms in TCP Peachtree outperform the TCP NewReno when combined with our hierarchical groups and ACK filter. It is also shown that TCP Peachtree can have very good scalability.**

I. Introduction

Satellite networks will play a crucial role in the global infrastructure of the Internet. They do not only provide global coverage, but also capable of sustaining high bandwidth levels and supporting flexible and scalable network configurations. Multicasting provides an efficient way of disseminating data from a sender to a group of receivers. A large variety of reliable multicast protocols [6], [10], [12], [13] have been proposed for the terrestrial wirelined networks where most of losses occur due to congestion. Satellite networks have high *bit error rate* (BER), long propagation delays and asymmetrical bandwidth [1], which result in clearly different characteristics for satellite multicasting compared to multicasting in terrestrial wirelined networks. In addition to the *acknowledgment implosion* and *scalability problems* appeared in terrestrial wirelined networks, satellite multicasting has the following additional problems: *low bandwidth feedback Link*, *different multicast topology* and *congestion control problem*. In the satellite domain, relatively few research work has been done on reliable multicast protocols in the past [8], [9]. In this paper, we present *TCP-Peachtree* to solve the multicast problems mentioned above. TCP-Peachtree includes the following parts: *multicast group formation*, *local error recovery*, *ACK aggregations*, and *congestion control*. Our contributions are as follows: Consider the special satellite multicast topology, we use a B+ tree hierarchy to form dynamic multicast groups for *ACK aggregation* and *local recovery*; Due to the long propagation delay in satellite multicast, we use *Jump Start* to improve the start behavior of TCP; In order to address high packet loss rate in satellite multicast, we use *Quick Recovery* to recover multiple packet losses in one window of data.

This work is supported by NASA-GRC under Project Number: NAG3-2585.

The paper is organized as follows: the multicast procedures in TCP Peachtree are presented in Section 2. Then the congestion control problem is discussed in Section 3. Simulation results are given in Section 4. Conclusions are presented in Section 5.

II. Multicast Procedures in TCP-Peachtree

A. The B+ tree hierarchy

B+ tree is a data structure typically used for searching data with data pointers stored only at the leaf nodes of the tree. Here, we use B+ tree to update hierarchical structure automatically instead of searching for a key. So we modify the traditional B+ tree as follows:

- The modified tree forms a multicast group.
- Each leaf node corresponds to a multicast subgroup which consists of physical receivers.
- Each internal node corresponds to a logical subgroup, which is created by choosing the DR from each of its child subgroups.
- In a subgroup, one member is selected as the *Designated Receiver* (DR), which is responsible for *ACK aggregation, local error recovery*, and *exchanging information* with members in its parent subgroup on behalf of all members in its subgroup.
- Each node (i.e., a multicast group) has at most M members.
- Each node (i.e., a multicast group), except the root, has at least $\lceil M/2 \rceil$ members. The root node has at least two members if it is an internal node.

0-7803-7632-3/02/$17.00 ©2002 IEEE

B. Members Joining a Multicast Group

A new member sends a particular $LOOK\_FOR\_DR$ packet with *time-to-live* (TTL) parameter equal to 1. Upon receiving this packet, the DRs send IM_DR type of ACK to the new member, which then chooses the DR with the smallest RTT value. If no reply comes back from any DR, the TTL value is increased by one and this procedure is repeated until a DR is found with the RTT value satisfying: $RTT/2 < TRTT$, where $TRTT$ is a threshold, and joins that multicast group, or stops if no DR can be found with a very large TTL value. In the latter case, the new member must initiate a new multicast group by sending a signalling message to the sender and becomes the DR of this multicast group initially. When a new member joins a multicast group, it must first join a subgroup at the leaf in the hierarchical structure. If the number of the members in a multicast subgroup exceeds a given threshold M, then we split the group into two subgroups, with one subgroup keeping the current DR while the other subgroup selecting a new DR, which is selected dependent on *packet loss statistics*. As a result, a new DR in the upper logical multicast is selected. We repeat this procedure in the upper logical multicast groups if necessary until the highest logical multicast group is reached.

C. Members Leaving a Multicast Group

When a member leaves a multicast group and the number of group members becomes less than $\lceil M/2 \rceil$, then we redistribute group members to a neighboring group, or merge two groups into one. In the latter case, we need to identify a DR for the new subgroup and remove the according DR from the upper logical multicast group. We repeat this procedure if necessary until the highest logical multicast group is reached. If the last member in the multicast group is removed, it also sends a release message to the sender to inform it that this multicast group is removed.

D. ACK Aggregation

The source multicasts packets directly to all receivers. Received packets are reported in ACKs. The DR in a logical multicast group receives ACKs from members in its group. After it gets all ACKs for a packet from all members in its group, the DR sends an ACK to the upper logical subgroup. The DR in the upper logical group also receives ACKs from its members and sends ACKs to its parent logical group. The DR in the highest logical group then sends an ACK to the satellite via the uplink channel. So for each packet only one ACK is sent to the source and the feedback implosion problem is solved consequently. If a receiver never acknowledges a packet, it will be dropped from the multicast group by the DR.

E. Local Error Recovery

Each DR in a subgroup maintains a buffer to hold a number of packets for *local retransmission*. The missing packets are reported in NAKs. When a member does not receive a packet correctly, it sends a NAK to the DR in its subgroup. If the lost packet is in the DR's buffer, the DR will retransmit the lost packet to the receiver immediately. If not, the DR first multicast a NAK to all members in this subgroup. Any member having the correct packet in its buffer can unicast that packet to the DR. In the meantime, the DR will set a timer for the lost packet, if the lost packet is received from other members in this subgroup, the timer will be canceled and the DR will unicast the lost packet to the corresponding receiver if only one NAK is received for the lost packet, or multicast the lost packet to the whole subgroup if multiple NAKs for this lost packet are received. If a timeout occurs, the DR will send a NAK to the DR in its parent logical group and will try to get the lost packet from its upper logical subgroup. This procedure is repeated until the highest logical subgroup is reached. In such a case, a NAK is sent to the satellite via the uplink channel. The source will then multicast the missed packets to all receivers.

III. Congestion Control in TCP-Peachtree

TCP-Peachtree congestion control contains two new algorithms: *Jump Start* and *Quick Recovery* as well as the two traditional TCP algorithms, *Congestion Avoidance* and *Fast Retransmit*. The TCP SACK option is also adopted. Moveover, an *ACK filter* is introduced.

A. NIL Segments

NIL segments are low priority segments that do not carry any new information. If a router on the connection path is congested, then it discards the *NIL segments* first. Consequently, the transmission of *NIL segments* does not cause a decrease of throughput of actual data segments, i.e, the traditional segments. If the routers are not congested, then the *NIL segments* can reach the receiver. The sender sets one or more of the six unused bits in the TCP header to distinguish *NIL segments* from data segments. Therefore, the receiver can recognize the *NIL segments* and for each of them transmits an ACK back to the sender. The ACKs for *NIL segments* are also marked using one or more of the six unused bits of the TCP header and are carried by low priority IP segments. Upon receiving ACKs for NIL segments, the sender interprets those ACKs as the evidence that there are unused resources in the network and accordingly, can increase its transmission rate. In TCP-Peach [2], a type of low priority segments, called *dummy segments* to probe the availability of network resources. In TCP-Peachtree,

we use *NIL segments* to probe the availability of network resources and also for error recovery.

B. The ACK Filter

The *ACK filter* is designed to aggregate ACKs from multiple return paths into one ACK. Whenever an ACK is received by the sender, it first goes through the ACK filter in order to eliminate duplicate ACKs from different return paths. The sender keeps the multicast group IDs in memory and used them for ACK filtering. There are three types of ACKS, i.e., NAK, ACK and NIL ACK, which are treated differently.

C. The Jump Start Algorithm

TCP *Slow Start* algorithm is disadvantageous in networks with long propagation delays. In TCP-Peachtree, NIL segments described in III-A are used to improve the *Slow Start* Algorithm. The basic idea of *Jump Start* is that in the beginning of a connection the TCP sender sets the congestion window, *cwnd* to 1 and after the first data segment, it transmits $(rwnd - 1)$ *NIL segments* every $\tau = RTT/rwnd$. As a result, after one *round trip time*, the congestion window size *cwnd* increases very quickly. Note that the TCP sender can estimate RTT during the connection setup phase. Here RTT is the largest value the sender gets from the receivers.

D. The Quick Recovery Algorithm

The *Quick Recovery* substitutes the classical Fast Recovery algorithm [7] with the objective of recovering multiple losses in a window and solving the throughput degradation problem due to link errors. When one or more segment losses are detected by either 3 duplicated SACKs or a NAK, we use the *Fast Retransmit* Algorithm to transmit a missing packet. After completing the *Fast Retransmit* Algorithm we apply the *Quick Recovery* algorithm. Like the TCP SACK proposed in [4], *Quick Recovery* maintains a data structure called *scoreboard* to update information about missing packets and a variable called *pipe* that represents the estimated number of packets outstanding in the path. The sender always sends the missing packets first. If no missing packet exists, the sender sends a new packet. Whenever a SACK is accepted, the *retransmit timer* is also reset. The sender exits *Quick Recovery* when a recovery ACK is received ACKing all data that was outstanding when *Quick Recovery* was entered. When packet losses are detected, the *QuickReco very* behaves conservatively, i.e., the sender halves its congestion window, *cwnd*. The *Quick Recovery* will be terminated roughly within one RTT time, NIL segments sent during *Quick Recovery* will be received in the *Congestion Avoidance* phase and the congestion window, *cwnd*, will be increased by one for each NIL ACK. As a result, the number of allowed NIL segments is set as *cwnd* so that the congestion window becomes the original value before the *Quick Recovery* very rapidly.

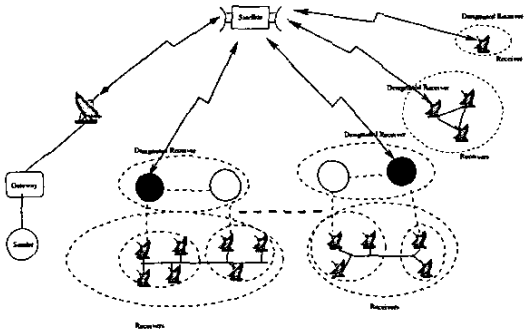

Fig. 1. The Simulation Scenario for TCP Peachtree Multicast

IV. Performance Evaluation

A. Simulation Scenario

We consider the following satellite multicast scenario as shown in Figure 1: There is a source sending packets to the satellite through a gateway, then the satellite multicasts packets to all receivers. Some receivers may have terrestrial connection among them, but not all of them need to be connected by terrestrial networks. The number of receivers is not fixed and can be very large. The receivers use the satellite uplink channel as the feedback link to the sender. Different receivers may experience different channel conditions at the same time and also the channel conditions are time-variant. Since the GEO satellite network is considered, the RTT values $(550ms)$ from the source to the receivers are approximately the same. There are background unicast connections from the gateway to the receivers. Congestion may occur in the gateway that connects the source and the satellite. The measured packet loss probability due to link errors in the channel varies from 10^{-6} to 10^{-2}. Furthermore, we assume the gateway buffer length to be 50 segments. $rwnd = 64$ segments. We also assume that the link capacity is $c = 1300$ segments/sec which is approximately 10 Mb/sec for TCP segments of 1000 bytes. Moveover, there are 11 unicast TCP NewReno connections from the gateway to the receivers. For those connections, we assume $rwnd = 64$ segments.

B. The Loss Path multiplicity Problem

In satellite multicasting, there are multiple paths to the receivers via the downlink channel. Multiple paths can cause *the*

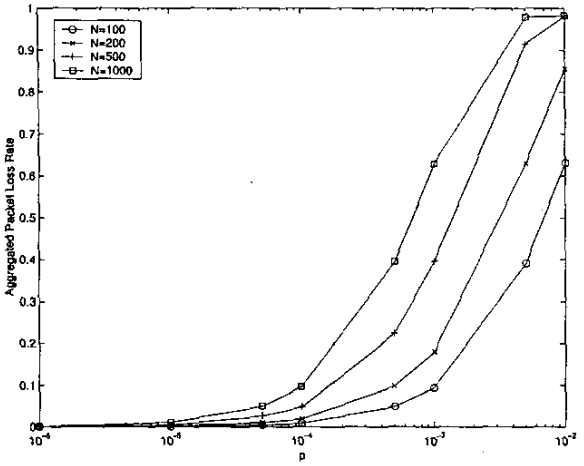

Fig. 2. The Aggregated Packet Loss Rate from the satellite to the receivers vs. p for $TRTT = 30ms$ and $M = 10$

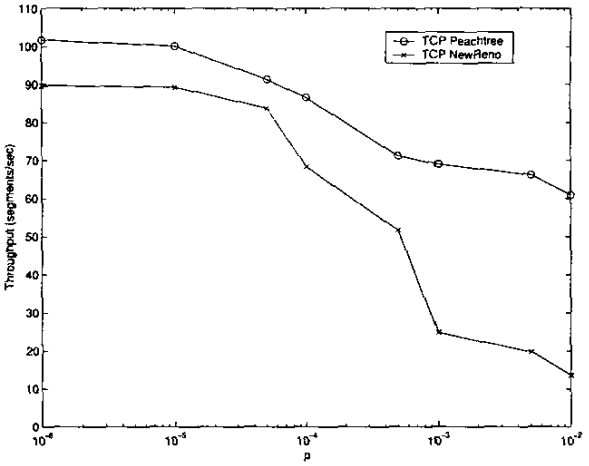

Fig. 3. Throughput Performance Comparison of TCP Peachtree and TCP NewReno for different values of p for $TRTT = 30ms$ and $M = 10$

loss path multiplicity problem [3]. Consider the satellite multicasting scenario as shown in Figure 1. Let p_i be the loss probability for a packet from the satellite to the $i-th$ receiver, where $0 < p_i \ll 1$ and it might be different for different receivers due to channel conditions and other factors. Let $p_{aggregate}$ be the aggregated loss probability for a packet from the satellite to the receivers, then

$$p_{aggregate} = 1 - \prod_{i=1}^{N}(1 - p_i) \qquad (1)$$

As $N \to \infty$, $p_{aggregate} \to 1$, i.e., as the number of loss paths increases, the loss probability for a packet also increases.

Here we assume all p_i's are equal, $p_i = p$. The aggregated packet loss rate from the satellite to the receivers is obtained as shown in Figure 2 with N=100, 200, 500 and 1000 respectively, where N is the total number of receivers in the multicast session.

Figure 2 clearly demonstrates the loss path multiplicity problem, i.e., even if the packet loss rate p from satellite to a single receiver is rather small, with increasing number of receivers, the packet loss rate is approaching 1. Although the aggregated packet loss rate is rather high, the multicast link errors have the property that different receivers may have different channel conditions so that some receivers receive the packet while the others do not. In other words, the lost packet may be received by some receivers. This makes it possible to use the local recovery schemes to solve the loss path multiplicity problem. In TCP Peachtree, a hierarchical multicast group is designed to recover this type of errors locally.

C. Throughput Performance Comparison with TCP NewReno

TCP NewReno [5] is a TCP unicast algorithm, the most important feature of TCP NewReno is that it can recover multiple missing packets in one window of data. TCP Peachtree congestion algorithm discussed in section III is also designed to recover multiple packets in one window of data. In order to evaluate TCP Peachtree congestion algorithms, we use TCP NewReno only for congestion control in our simulations and keep the hierarchical groups and the ACK filter as we introduced in section II and III-B. Thus, we can compare the throughput performance between TCP Peachtree and TCP NewReno. For multicast applications, we assume $N = 200$, $TRTT = 30ms$ and $M = 10$, the resulting throughputs are shown in Figure 3 for different values of p.

Figure 3 shows that the throughput performance of TCP Peachtree is much better than the TCP NewReno, especially when p is large, i.e., TCP Peachtree is more suitable for reliable satellite multicast.

D. Scalability

Scalability is one of the most important metrics of IP multicast schemes. Usually,scal ability is measured by the performance over different numbers of receivers in a multicast session. Schemes with good scalability can be applied to very large size of receivers. Here we use throughput as the performance measurement. For $TRTT = 30ms$ and $M = 10$, the resulting scalability is shown in Figure 4, where we observe that the throughput is constant and does not decrease with N increasing from 50 to 1000. We can conclude that TCP

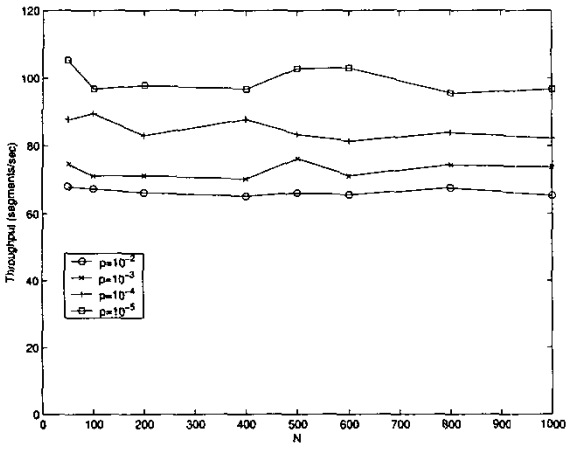

Fig. 4. Scalability for $TRTT = 30ms$ and $M = 10$

Peachtree has good scalability. The reasons are based on the facts:

- The *acknowledgment implosion problem* is solved by the ACK fusion procedure performed by the DRs of the hierarchical multicast groups and the ACK filter on the sender side.
- The *loss path multiplicity problem* can be solved by *local error recovery* and *NIL recovery*.
- The congestion control algorithms in TCP Peachtree can recover multiple packet losses in one window of data so that TCP Peachtree is very suitable to work in applications with high packet loss rates due to link errors.

V. Conclusions

In this paper, the TCP Peachtree is proposed for reliable multicast in satellite IP networks. TCP Peachtree uses a modified B+ tree-like hierarchical multicast group to solve the acknowledgment implosion and scalability problems in reliable IP multicast applications. Two new congestion control algorithms are also presented, i.e., *Jump Start* and *Quick Recovery*, so that TCP Peach is suitable for satellite IP networks with long propagation delays and high bit error rates. NIL segments are used to exploit the availability of network resources and recover lost packets on receiver side. Simulation results show that the congestion control algorithms in TCP Peachtree perform better than that of the TCP NewReno. It is also shown that TCP Peachtree has very good scalability.

References

[1] I. F. Akyildiz, G. Morabito, S. Palazzo, "Research Issues for Transport Protocols in Satellite IP Networks," *IEEE Personal Communications*, Vol. 8, No. 3, pp. 44-48, June 2001.

[2] I. F. Akyildiz, G. Morabito, S. Palazzo, "TCP Peach: A New Congestion Control Scheme for Satellite IP Networks," *IEEE/ACM Transactions on Networking*, June 2001, 307-321.

[3] S. Bhattacharyya, D. Towsley and J. Kurose, "The Loss Path Multiplicity Problem in Multicast Congestion Control," *Proc. of IEEE INFOCOM'99*, New York, NY,Ma rch 1999.

[4] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communication Review*, Vol. 26, pp. 5-21, July 1996.

[5] S. Floyd, and T.H enderson, "The NewReno Modification to TCP's Fast Recovery Algorithm", *RFC 2582*,A pril 1999.

[6] S. Floyd, V. Jacobson, and C. G. Liu, "A Reliable Multicast Framework for Light Weight Session and Application Level Framing," *in Proc. of ACM SIGCOMM'95*, pp. 342–356, August 1995.

[7] V. Jacobson, "Congestion Avoidance and Control," *Technical Report*, April 1990.

[8] M. Jung, J. Nonnenmacher, E. W. Biersack, "Reliable Multicast via Satellite: Uni-directional vs. Bi-directional Communication,"*Pr oc. of KiVS'99*, Darmstadt, Germany,Ma rch 1999.

[9] M. Koyabe and G. Fairhurst, "Reliable Multicast via Satellite: A Comparison Survey and Taxonomy," *International Journal of Satellite Communications*, Vol. 19, Issue 1, January/February 2001, pp. 3-28.

[10] L. H. Lehman, S. J. Garland, and D. L. Tennenhouse, "Active Reliable Multicast," *IEEE INFOCOM'98*, San Francisco, CA, March, 1998.

[11] M. Marchese,"Performance Analysis of the TCP Behavior in a GEO Satellite Environment," *Computer Communications Journal*, Vol. 24, pp. 877-888, 2001.

[12] S. Paul and K. K. Sabnani, "Reliable Multicast Transport Protocol (RMTP)," *IEEE Journal of Selected Areas in Communications*, vol. 15, no. 3, pp. 407–421, April 1997.

[13] I. Rhee, N. Balaguru and G. N. Rouskas, "MTCP: Scalable TCP-like Congestion Control for Reliable Multicast," *Proc. of IEEE INFOCOM'99*, New Tork, NY, March 1999.